Uzzy Proudly Presents

TIPURI DE STRUCTURI DE DATE SI APLICATII

STRUCTURI DINAMICE DE DATE

O mare importanta în programarea si utilizarea calculatoarelor o are modul în care se face organizarea datelor. De cele mai multe ori , în aplicatii , datele se prezinta sub forma unor multimi sau colectii , iar prelucrarea lor nu poate fi conceputa fara o organizare corespunzatoare.

Organizarea datelor este un proces complex în care stabilirea structurilor logice si fizice reprezinta elemente fundamentale de care depinde, în mare masura, însasi eficienta prelucrarii.

O data care apare ca o entitate indivizibila, atât în raport cu informatia pe care o reprezinta, cât si în raport cu procesorul care o prelucreaza, se numeste ***data elementara*** sau ***scalara.***

O colectie de date pe care s-a definit o structura si pentru care s-au stabilit procedee de înregistrare si identificare a componentelor se numeste ***structura de date***.

Componentele unei structuri de date pot fi date elementare sau alte structuri de date care sunt identificate si selectate prin nume (identificatori), sau prin pozitia pe care o ocupa în structura.

Un ***tip de date*** defineste atât o multime de valori cât si o multime de operatii ce se pot efectua cu aceste valori. Fiecarei date i se asociaza un tip unic. Astfel, datele ce pot fi prelucrate în programele scrise în limbajul Turbo Pascal pot fi clasificate astfel:

1. Tipuri de date simple (elementare, scalare) :
   1.1. Tipuri de date predefinite:
      1.1.1. Întregi
      1.1.2. Reale
      1.1.3. Booleene
      1.1.4. Caracter
   1.2. Tipul de date enumerare
   1.3. Tipul de date subdomeniu (interval)
2. Tipuri de date compuse (structurate) :
   2.1. Tipul de date tablou (array)
   2.2. Tipul de date sir de caractere (string)
   2.3. Tipul de date multime (set)
   2.4. Tipul de date înregistrare (record)
   2.5. Tipul de date fisier (file)
3. Tipul de date pointer
4. Tipul procedural

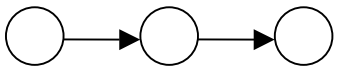
Tipurile de date structurate sunt agregari ale unor tipuri deja definite, simple sau structurate, standard sau definite de utilizator. Deci, din punct de vedere al complexitatii lor, datele se pot clasifica în *date elementare* si *date structurate*. O alta clasificare a datelor poate fi facuta d.p.d.v. al alocarii memoriei. Acestea pot fi grupate în *date alocate static* si *date alocate dinamic*. Procesul de alocare dinamica a memoriei permite ca în timpul executiei programului sa poata fi create, utilizate si distruse anumite variabile numite variabile dinamice.

Structurile dinamice de date sunt structuri de date al caror numar de componente se modifica în timpul executarii programului (dimensiunea programului creste si/sau descreste).

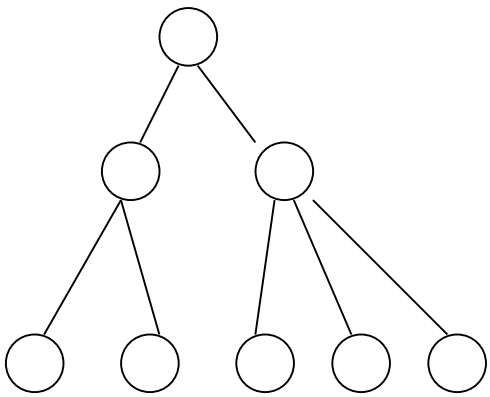
Structurile dinamice sunt mult mai flexibile decât cele statice si extrem de avantajoase acolo unde este necesara aceasta flexibilitate.

O structura dinamica poate avea toate elementele structurate pe un singur nivel, caz în care se numeste *lista liniara*, poate avea elementele structurate pe mai multe niveluri, caz în care se numeste *arbore*, sau poate cuprinde elemente care nu pot fi grupate pe niveluri, putând exista legaturi între oricare elemente, structura numindu-se în acest caz *retea* sau *graf*.

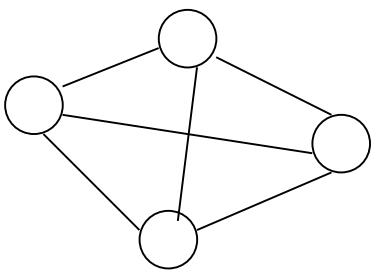
LISTA



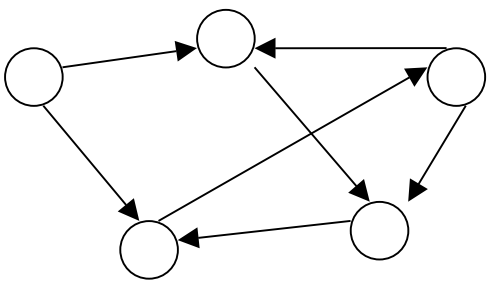
ARBORE



GRAFURI NEORIENTATE



GRAFURI ORIENTATE



Am ales ca tema pentru aceasta lucrare ***structurile dinamice de date*** deoarece ele Ósi gasesc utilitatea Óntr-un mare numar de domenii. At, t listele c, t si grafurile (acestea incluz, nd si arborii) sunt folosite Óna foarte multe aplicatii.

Grigore Moisil spunea : î *Azi teoria grafurilor a devenit o disciplina majora, desi nu-si gaseste locul Óntr-o clasificare dogmatica a capitolelor matematicii. Folosirea teoriei grafurilor Óna domenii variate, de la chimie la economie, de la studiul retelelor electrice la critica textelor si la politica, Óî dau azi un prestigiu de care cel ce clasifica stiintele trebuie sa tina seama.î*

Listele sunt utilizate Óna mai multe domenii de activitate, folosirea lor fiind indispensabila: liste cu personalul unei Óntreprinderi, liste cu materiale cumparate sau produse v, ndute, cererile catre o anumita informatie ,Óntr-o retea computerizata de baze de date, sunt memorate Óntr-o lista, scanarea memoriei pentru cautarea unor virusi poate fi tratata ca si cautarea unei secvente Óntr-o lista etc.

Arborii sunt utilizati si ei Óna multe domenii: chimie organica, fizica etc . Astfel, ei pot fi utilizati Óna diverse clasificari , Óna sortari etc.

Structurile dinamice de date ofera elevilor posibilitatea de a rezolva un mare numar de probleme cu aplicatii practice.

Œa aceasta lucrare vor fi prezentate listele, arborii si grafurile orientate si neorientate.

La **Liste** ñ vor fi date , pentru toate tipurile de liste, definitiile, reprezentarile grafice, operatiile cu liste (crearea, stergerea, parcurgerea, adaugarea), procedurile respective (pentru un exemplu dat) si aplicatii cu fiecare tip de lista.

La **Grafurile neorientate -** se vor defini notiunile necesare Óna rezolvarea problemelor ce utilizeaza grafurile , se vor trata reprezentarea grafurilor neorientate, parcurgerea lor si se vor prezenta c, teva aplicatii .

La **Arbori ñ** se vor defini notiunile ce vor fi utilizate si se vor trata Óna special arborii binari si problema arborelui partial minim.

La **Grafurile orientate -** se vor defini noile notiuni, se va trata reprezentarea grafurilor orientate si probleme de drumuri Óna grafuri. Se vor trate algoritmii de baza: Dijkstra, Roy-Floyd, Roy-Warshall si metoda drumului citic Óntr-un graf de activitati.

Œa capitolul cu aplicatii va fi prezentat si un program de desenare a arborilor binari cu un numar fixat de v, rfuri.

CONSIDERATII METODICE

Structurile dinamice de date ridica de multe ori probleme elevilor. Aceste probleme apar din mai multe motive. Unul dintre aceste motive se datoreaza neÓntelegerii corecte a relatiei dintre variabila dinamica si variabila de tip reper (referinta) atasata variabilei dinamice. O alta greseala pe care o fac frecvent elevii este confundarea operatiilor specifice fiecarui tip de lista. Probleme apar si la Ósusirea metodelor de rezolvare a problemelor care necesita Ó modelare grafuri.

Astfel de situatii pot fi Ósa evitate prin aplicarea de catre profesor a unor principii ale procesului de Óvatam, nt, precum si utilizarea unor metode si mijloace de Óvatam, nt adecvate.

Procesul de Óvatam, nt este un proces formativ-educativ care influenteaza dezvoltarea personalitatii si este o cerinta actuala de pregatire a elevului Ó vederea integrarii sociale, a afirmarii personalitatii fiecaruia. Una dintre cele mai importante conditii pentru reusita predarii Ó scoala a unei discipline o constituie structurarea, constientizarea si ierarhizarea unor obiective generale si specifice adaptate particularitatilor de v, rsta ale elevilor, continutului cunostintelor precum si pregatirea stiintifica si metodica a profesorilor.

Profesorul de informatica trebuie sa valorifice potentialul formativ-creativ al disciplinei, posibilitatea acesteia de a structura g, ndirea, de a dezvolta flexibilitatea ei, de a forma deprinderi si atitudini conform continutului de idei.

In Óvatam, ntul informatic se aplica mai multe *principii ale procesului de Óvatam, nt*, printre care:

- Principiul participarii active si constiente a elevilor la procesul de Óvatam, nt
- Principiul sistematizarii si continuitatii
- Principiul accesibilitatii cunostintelor si deprinderilor
- Principiul intuitiei
- Principiul pasilor mici
- Principiul progresului gradat al performantelor Ó pregatire
- Principiul conexiunii inverse(feed back-ului).

Profesorului de informatica Ó revine misiunea de a realiza ore cu un grad mare de eficienta. Pentru aceasta el va trebui ca, tin, nd cont de principiile de mai sus , sa realizeze o proiectare corecta a tuturor actiunilor si sa realizeze aplicarea lor eficienta Ó activitatea de predare-Óvatare propriu-zisa. Acest lucru se realizeaza prin proiectarea didactica prin care vor fi stabilite obiectivele generale, care vor fi conforme cu continutul programei scolare si a obiectivelor specifice.

Pentru o proiectare metodologica eficienta, odata cu stabilirea obiectivelor generale si specifice, profesorul trebuie sa aleaga si sa utilizeze metode de activizare a elevilor, sa aplice tratarea diferentiata, dezvoltarea motivatiei Óvatarii si sa realizeze prelucrarea calitativa a cunostintelor predate.

Din ansamblul mijloacelor si metodelor de Ónvatam, nt, pentru cazul lectiei de dob, ndire de cunostinte se pot utiliza urmatoarele:

- ***Metode***
  - ***ñde comunicare***
    - orale:
      - expunerea,
      - explicatia
    - conversative:
      - conversatia
      - problematizarea
      - discutia colectiva
  - ***de actiune***
    - ~ prin aplicatii specifice temei
    - ~ instruire prin activitati independente

Mijloace (resurse materiale):

- Planse didactice
- Seturi de aplicatii
- Fise de lucru pentru elevi
- Culegeri de probleme
- Proiect didactic

Forma de participare este cea colectiva.

Astfel o lectie poate avea urmatorul ***mod de desfasurare*** Ón conformitate cu obiectivele propuse:

-un moment important al lectiei este *organizarea clasei* - moment al lectiei care are ca obiectiv operational crearea starii psihologice favorabile desfasurarii lectiei. Œn aceasta etapa profesorul desfasoara o activitate de natura organizatorica: noteaza absente, asigura disciplina, pregateste materialele necesare, Ón timp ce elevii Ósi pregatesc materialele si desfasoara o actiune de autoorganizare si control propriu. Se urmareste captarea atentiei elevilor;

-un alt moment al lectiei este- *anuntarea subiectului lectiei* -moment Ón care profesorul scrie pe tabla titlul lectiei. Comunicarea obiectivelor lectiei nu este importanta, fiind suficient ca aceste obiective sa fie clare pentru profesor, care va face pentru sine, dupa lectie, o analiza a atingerii lor;

-urmeaza- *procesul de predare-Ónvatare* -ce presupune comunicarea cunostintelor si dirijarea Ónvatarii. Sunt reactualizate notiunile necesare si se comunica noile cunostinte, Ón timp ce elevii noteaza si analizeaza. Prezentarea continutului nou este bine sa se faca prin activizarea elevilor, prin rezolvarea unor sarcini de lucru concrete, care sa duca la descoperirea de catre ei Ónsisi a noilor continuturi;

-urmatoarele etape sunt- *fixarea si consolidarea cunostintelor* si *indicarea temei.* Se urmareste obtinerea performantei si sa se asigure feed-back-ul.

La sf, rsitul orei se poate face o evaluare sumativa pe baza unor teste orale cu ßntrebari . *Evaluarea rezultatelor* este un moment important al lectiei si consta ßn notare, aprecieri ale profesorului sau autoaprecieri, rezultate la teste.

Pregatirea riguroasa a activitatii didactice este foarte importanta pentru asigurarea eficientei orei de curs. Planurile de lectie , schematice sau detaliate, sunt instrumente de lucru absolut necesare. ßn elaborarea unui plan de lectie se evidentiaza creativitatea, imaginatia, talentul pedagogic al profesorului, neexist, nd scheme prestabilite, ci factori variabili care intervin, impun, nd variante diferite.

Prezentam un **model orientativ al planului de lectie:**

I. Date generale:

- Obiectul;
- Clasa;
- Data;
- Subiectul lectiei;
- Scopul lectiei;
- Tipul de lectie;
- Obiective perfomative(operationale, cognitive si afective)
- Metode si procedee didactice;
- Mijloace de ßnvatam, nt, material didactic;
- Bibliografie;

I. **II. Scenariul didactic:**

Aceasta parte a planului de lectie poate fi:

* **Detaliata** , stabilind:

- Volumul de cunostinte;
- Activitatea profesorului si a elevilor;
- ßntrebarile profesorului;
- Exercitiile de rezolvat;
- Schemele care vor fi prezentate;
- Metodele si procedeele utilizate;
- Materialul didactic folosit;
- Etapele lectiei;
- Timpul acordat fiecarei secvente din lectie;
- Modalitatea de evaluare a activitatii.

* **Schematica**, cuprinz, nd:

- Etapele lectiei;
- Problemele mai importante

Obiectivele generale pentru structurile dinamice de date (conforme cu continutul programei scolare) sunt urmatoarele:

- Fixarea notiunilor de structuri de date
- Fixarea notiunilor de alocare statica si alocare dinamica
- Formarea deprinderilor de identificare si rezolvare algoritmica a problemelor care necesita ón modelare liste liniare ólantuite
- Formarea deprinderilor de identificare si rezolvare algoritmica a problemelor care necesita ón modelare grafuri neorientate, respectiv grafuri orientate
- Cunoasterea de catre elevi a avantajelor/dezavantajelor diverselor reprezentari ale grafurilor
- Formarea capacitatii de a alege din mai multe metode de rezolvare a unei anumite probleme pe cea optima, ón functie de specificul problemei si de resursele calculatorului.
- Insusirea de catre elevi a modului de lucru cu structuri arborescente

Vor fi stabilite at,t obiectivele specifice fiecarui capitol ón parte, c,t si obiectivele educationale (cognitive, psihomotorii si afective). Se va urmarii realizarea tuturor obiectivelor propuse, ón cele mai bune conditii, prin utilizarea unor metode si mijloace adecvate .

Evaluarea rezultatelor scolare este o activitate centrala a unui proces de instruire si óvatare. Ea nu poate fi saparata de instruire si de óvatare.

Evaluarea trebuie sa aiba obiective clar definite, metode si tehnici eficiente si moderne, metode de investigare si comunicare a rezultatelor scolare pentru fiecare elev.

O evaluare eficienta trebuie sa arate profesorilor daca au fost atinse obiectivele curriculare, sa-i ajute pe acestia sa faca o diagnoza a progresului elevilor si sa adapteze actiunile elevilor cu posibilitatile acestora. De asemenea ea trebuie sa orienteze elevii ón alegerea celei mai bune cai de afirmare, sa ajute profesorii sa-si evalueze propria activitate si sa furnizeze feed-back-ul catre parinti.

Evaluarea poate fi:

- Predictiva (initiala)
- Formativa (continua)
- Sumativa (finala)

Exista mai multe instrumente de evaluare , profesorul put,nd alege ceea ce crede ca este mai indicat. Astfel sunt instrumente:

- Traditionale :
    - probe scrise
    - probe orale
    - probe practice

- Alternative:
    - Probe scrise
    - Probe orale
    - Probe practice
    - Observarea directa a elevului în timpul activitatii
    - Investigatia
    - Proiectul de cercetare
    - Portofoliul
    - Tema pentru acasa
    - Autoevaluare

în Legea Învatam, ntului se precizeaza urmatoarele :

ì Idealul eductional al scolii rom, nesti consta în dezvoltarea libera, integrala si armonioasa a individualitatii umane, în formarea personalitatii autonome si creative.î

ì Învatam, ntul are ca finalitate formarea personalitatii umane.î

CAP. I

LISTELE

CONSIDERATII METODICE

Listele se studiaza în clasa a X-a la obiectele: ìProgramarea calculatoarelorî si la ìAplicatii practice de laboratorî. Tema face parte din capitolul ìStructuri de date înlantuiteî si urmeaza dupa capitolul ìAlocare dinamicaî.

Pentru studierea acestei teme , elevilor trebuie sa li se reaminteasca notiuni legate de : tipul record, alocare dinamica, tablouri.

Pentru a expune elevilor utilitatea acestei structuri de date este nevoie de a enumera unele domenii de activitate în care folosirea listelor este indispensabila: liste cu materiale , lista candidatilor la un examen de admitere, liste cu personalul dintr-o întreprindere etc.

Referitor la construirea si exploatarea listelor, trebuie explicat elevilor ca acestea pot fi implementate folosind doua modalitati relativ distincte, prezent,nd, fiecare în parte, avantaje , dezavantaje si mijloace proprii de tratare a operatiilor specifice.

Tratarea listelor se poate face folosind tablouri statice sau structuri dinamice de date înlantuite. Elevii trebuie sa înteleaga, în primul r,nd, modalitatea de a alege , pentru rezolvarea unei probleme, una dintre cele doua implementari ale listelor. Se va preciza ca tablourile statice permit crearea unei liste cu un numar de componente limitat de dimensiunea tabloului, a memoriei calculatorului sau a spatiului de pe disc(discheta).

Crearea unei liste folosind alocarea dinamica a memoriei presupune o structura mult mai flexibila, dar relativ mai greu de exploatat, limitata de memoria calculatorului sau spatiul pe disc(discheta).

Deci tablourile se vor folosi pentru acele liste în care se cunoaste numarul maxim de componente, pe c,nd structurile dinamice de date se folosesc acolo unde nu se cunoaste de la început numarul componentelor.

Abordarea listelor prin prisma structurilor dinamice de date este o tema noua pentru elevi si , din acest motiv, trebuie tratata în mod gradat, de la notiuni mai simple spre un conglomerat de notiuni.

Se poate începe cu o expunere despre organizarea memoriei calculatorului, se va reaminti tipul referinta(reper) si modul cum poate fi alocata memoria pentru o variabila.

Elevii trebuie sa înteleaga motivatia utilizarii structurii dinamice si avantajele pe care le ofera folosirea acestora. Pentru aceasta li se poate propune o aplicatie pe care sa o rezolve cu ajutorul tablourilor. Dupa rezolvare le complicam putin problema, marind numarul componentelor , astfel înc,t sa depaseasca numarul maxim de elemente ale tabloului. Elevii vor descoperi astfel ideea de baza a structurilor dinamice- numarul componentelor variaza.

Fac, nd apel la intuitie si prin analogii cu alte exemple din viata cotidiana, profesorul trebuie sa determine elevii sa ßnteleaga notiunea de lista si sa deduca modalitatea de implementare a listelor cu ajutorul structurilor de tip referinta.

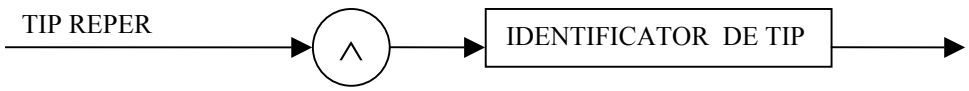
Pentru a usura ßntelegerea modului de memorare a listei se poate folosi o reprezentare grafica a acesteia, mention, nd ca fiecare element este caracterizat de informatia utila si de adresa urmatorului element.

Folosind principiile programarii structurate vom crea functii si proceduri pentru adaugarea unui element, parcurgerea unei liste, inserarea / stergerea unui element dintr-o anumita pozitie, cautarea unui element cu anumite caracteristici etc. Pentru deducerea algoritmilor se va folosi reprezentarea grafica a datelor.

Spre deosebire de datele alocate static, a caror alocare se realizeaza la ßnceput, ßn faza de compilare si se pastreaza pe durata ßntrgii executii a programului, datele alocate dinamic pot fi generate sau distruse ßn timpul executiei programului. Aspectul dinamic al structurii determina aparitia , ßn timp, a noi componente care trebuie legate ßn structura dinamica de celelalte componente. Pentru realizarea ßnlantuirii datelor, pe l, nga informatia propriu-zisa, fiecare componenta mai contine si o informatie suplimentara, numita informatie de legatura. Informatia de legatura cuprinde adresa componentei urmatoare din structura.

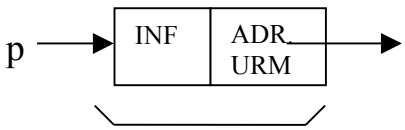
O variabila dinamica se aloca dinamic ßntr-o zona speciala numita HEAP (gramada) care este eliberata la ìdistrugerea ì variabilei dinamice. Neav, nd nume (nefiind declarate ßntr-o sectiune VAR) , variabilele dinamice vor fi referite prin intermediul altor variabile, numite , din acest motiv, variabile reper sau referinta.

Deci variabilele reper sunt alocate static si au ca valori adrese ale unor variabile dinamice de un anumit tip. Variabilelor dinamice li se pune ßn corespondenta un tip referinta ßn mod biunivoc: variabila reper contine referiri numai la variabila dinamica careia i-a fost pusa ßn corespondenta (contine adresa acesteia).
Tipul reper (referinta) se defineste astfel:



-unde prin identificator de tip s-a precizat tipul variabilelor dinamice pe care le refera variabila de tipul reper respectiv.
O variabila dinamica se apeleaza prin intermediul tipului reper corespunzator:

Daca variabila reper p are ca valoare adresa unei variabile dinamice, notata p^, spunem ca p refera variabila dinamica p^. Grafic acest lucru se reprezinta astfel:



p^

Crearea unei variabile dinamice presupune:

1) **alocarea unei zone de memorie** Ó HEAP pentru variabila dinamica
2) **initializarea zonei de memorie** corespunzatoare ei.

Alocarea memoriei se realizeaza prin apelul procedurii NEW definita Ó unit-ul System. Apelul procedurii:

NEW (p) -unde p este o variabila de tip reper.

Alocarea memoriei pentru o variabila dinamica nu presupune si initializarea ei, aceasta caz, nd Ó sarcina programatorului.

Eliberarea zonei de memorie ocupata de o variabila dinamica , alocata prin procedura NEW se realizeaza cu procedura DISPOSE , din unit-ul System.
Apelarea procedurii :

DISPOSE (p)

O ***lista liniara*** se poate defini ca o multime finita E de elemente , E={e_1,e_2,Ö .e_n} care satisface proprietatile:

1) exista o relatie de ordine Ó ntre elementele lui E astfel Ó nc, t oricare e_k are un predecesor si un succesor;
2) exista un element Ó n lista - elementul e_1 care nu are predecesor si care se numeste capul listei (prim);
3) exista un element Ó n lista ñ elementul e_n care nu are succesor si care se numeste element terminal(ultim).

Se numeste ***lista Ó lantuita*** o multime dinamica de structuri succesive de acelasi tip si care satisfac una sau mai multe relatii de ordine introduse prin pointeri.
Listele liniare pot fi de mai multe feluri:

- *lista liniara simplu Ó lantuita*- parcurgerea ei se face Ó ntr-un singur sens de la primul spre ultimul element, iar elementele pot fi inserate oriunde Ó n lista, sau pot fi sterse indiferent de pozitia lor Ó n lista;
- *lista liniara dublu Ó lantuita-* parcurgerea ei se poate realiza Ó n ambele sensuri, iar fiecare element contine, pe l, nga zona de informatie , doua zone de adresa: una pentru elementul anterior si una pentru elementul urmator;
- *lista circulara-*care poate fi simplu sau dublu Ó lantuita, ea obtin, ndu-se inchiz, nd o lista simplu, respectiv dublu Ó lantuita, prin legarea primului element al listei dupa ultimul element al ei.
- *stiva-* parcurgerea acestei liste se realizeaza Ó ncep, nd cu ultimul element introdus, iar operatiile de adaugare si de stergere se realizeaza la un singur capat al listei , numit v, rful stivei;
- *coada-* parcurgerea se realizeaza Ó ncep, nd cu primul element, adaugarea unui element se poate face numai la sf, rsitul listei , iar stergerea unui element se poate realiza numai la Ó nceputul listei.

Prelucrarea acestor structuri dinamice se realizeaza cu ajutorul variabilelor reper si a variabilelor dinamice.Trasaturile comune tuturor structurilor dinamice de date sunt:

- ele se dezvolta dinamic pornind de la o structura initial vida, toate programele de prelucrare vor trebui sa cuprinda secvente de ÓinceputÓ prin care sa se exprime faptul ca structura dinamica este vida;
- accesul la elementele structurii este asigurat prin intermediul adreselor de legatura; deoarece fiecare element cuprinde adresa elementului urmator,este necesar sa se cunoasca adresa primului element al structurii (aceasta adresa va fi pastrata ÓintrÓ-o variabila de tip reper asociata structurii).

Operatiile specifice tuturor structurilor dinamice sunt:

- *adaugarea unui element ÓinÓ structura* ñcare presupune:
  - *alocarea dinamica a memoriei pentru noul element*
  - *initializarea noului element alocat*
  - *legarea elementului ÓinÓ structura*

- *parcurgerea (traversarea) elementelor structurii-* se utilizeaza o variabila de lucru de tip reper ale carei valori reprezinta adresa elementului curent al structurii. Se va avea grija sa nu se distruga structura si sa nu se piarda adresa primului element al structurii.

- *stergerea unui element al structurii-* este posibila numai daca structura este nevida si presupune:
  - eliminarea elementului din structura prin modificarea adreselor de legatura *(stergere logica)*
  - eliberarea memoriei ocupate de elementul respectiv*(stergerea fizica)*

Pentru o mai usoara ńtelegere a listelor , este mai bine sa se ńceapa tratarea lor cu stiva , deoarece adaugarea si stergerea elementelor acestui tip special de lista se realizeaza la acelasi capat al listei , numit v,rful stivei, fapt care simplifica mult lucrurile.

Dupa ce elevii vor ńtelege modul de lucru cu acest tip de lista se poate trece la tratarea listelor de tip coada, acestea reprezent,nd o alta categorie speciala de liste liniare, ń care elementele se pot adauga numai la sf,rsitul listei, iar stergerea lor nu se poate face dec,t la ńceputul listei.

Se poate prezenta apoi modul de tratare a listelor de tip coada cu ajutorul unei santinele.

Lista liniara simplu ńlantuita poate ridica unele probleme, deoarece apar mai multe situatii at,t la adaugarea unui element ń lista c,t si la stergerea unui element din lista. Operatiile nu sunt dificil de realizat, ńsa deranjeaza faptul ca trebuie tratate at,tea cazuri particulare. Acest dezavantaj poate fi eliminat daca lista este creata cu doua santinele.

Lista liniara dublu ńlantuita nu ridica probleme daca elevii ńteleg modul ń care se leaga elementele atunci c,nd se cunosc adresele elementului precedent si urmator ale fiecarui element din lista. Existenta celor doua adrese de legatura simplifica at,t operatiile de adaugare c,t si pe cele de stergere. Parcurgerea listei se poate realiza ń ambele sensuri , fiind o lista simetrica.

Lista circulara va fi usor ńteleasa de catre elevi , daca acestia au ńteles modul de lucru cu listele simplu si dublu ńlantuite.

Pentru fiecare tip de lista se vor oferi exemple c,t mai simple, pentru ńceput, pentru ca, treptat, sa se rezolve probleme cu un grad din ce ń ce mai mare de dificultate.

Œn capitolul cu aplicatii se vor da exemple de probleme care folosesc ń rezolvarea lor tipurile de liste studiate.

<u>STIVA</u> (L I F O)

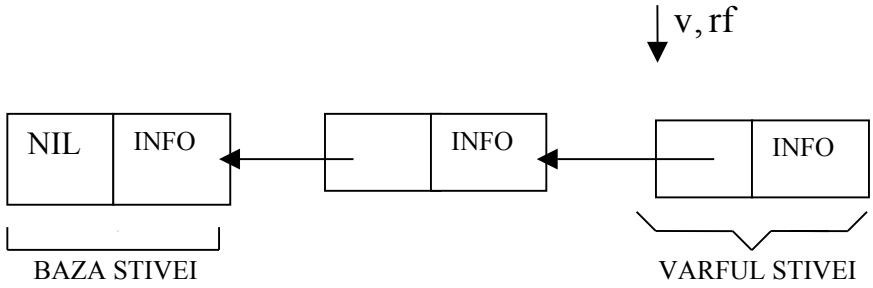
Stiva este o lista liniara de un tip special, în care adaugarea sau scoaterea unui element se face la un singur capat al listei, numit v,rful stivei.

Primul element introdus se numeste baza stivei. Informatia de legatura a fiecarui element din stiva reprezinta adresa elementului pus anterior în stiva, exceptie fac,nd baza , a carei informatie de legatura este NIL.

Pot fi oferite elevilor exemple de stive din viata cotidiana : o stiva de lemne , o stiva de farfurii , stive de carti etc.

Ei vor înelege astfel ca nu pot adauga un nou element (lemn, farfurie, carte) dec,t în v,rful stivei si nu vor putea sa scoata din stiva un element dec,t daca acesta se afla în v,rful stivei.

Reprezentarea grafica a unei stive:



Conditia de stiva vida este: v,rf=NIL

Operatii cu stive :

- ***Adaugarea unui element*** - este posibila numai în v,rful stivei si presupune urmatoarele etape:

  alocarea memoriei pentru noul element

  NEW (p)

  initializarea zonei de informatie din variabila dinamica ***p^*** **:**

  p^.info:=Ö .. sau Readln (p^.info)

  3. *legarea elementului în v, rful stivei* **:**

  p^.info := v,rf

  4. *mutarea v, rfului stivei:*

  v,rf := p

Ø ***Parcurgerea unei stive***- se vor parcurge elementele listei în ordinea inversa introducerii lor, dupa principiul : Last In ñ First Out.

Parcurgerea se realizeaza cu o variabila de tip reper *p* care contine adresa elementului curent :

se porneste din v,rful stivei :

p := v,rf

c,t timp stiva nu este vida (v,rf <> NIL) se parcurge elementul curent si se trece la urmatorul element din lista :

p := p^. next

Ø ***Stergerea unui element din stiva*** se poate sterge numai elementul din v,rful stivei (daca aceasta este nevida) :

1. se salveaza adresa v,rfului stivei într-o variabila reper p în vederea eliberarii memoriei:

   p:=v,rf

2. se muta v,rful stivei pe elementul urmator celui ce se va sterge :

   v,rf := v,rf ^.next

3. se elibereaza memoria ocupata de fostul v,rf :

   Dispose (p)

Consider,nd urmatoarele declaratii:

```
Type reper=^ element;
        element=record
                info:string[10];
                next:reper
        end;
Var  p, v, rf : reper;
        Text : string[10];
```

putem sa scriem urmatoarele subprograme:

Crearea stivei:

```
Procedure creare;
 Begin
        V, rf:=nil;
        Write (ë introduceti informatia :í);
        Readln ( text);
        While text<> ëí do
           Begin
                New(p);
                P^.info:=text;
                P^.next:=v, rf;
                V, rf:=p;
        Write(ë introduceti informatia :í);
                Readln(text);
           End;
 End;
```

Parcurgerea stivei:

```
Procedure parcurg;
  begin
        p:=v, rf;
        while p<>nil do
            begin
                writeln( p^.info);
                p:=p^.next
  end;
```

Adaugarea unui element in stiva:

```
Procedure adaug;
    begin
        new(p);
        write(ë introduceti informatia :í);
        readln( text );
        p^.info:=text;
        p^.next:=v, rf;
        v, rf:=p;
     end;
```

<u>Stergerea unui element din stiva:</u>

```
Procedure sterg;
   Begin
      p:=v, rf;
      v, rf:=p^.next;
      writeln(p^.info);
      dispose(p);
   End;
```

O problema simpla care ilustreaza modul de lucru cu stiva este cea care inverseaza caracterele unui cuv, nt citit de la tastatura dintr-o linie de intrare.

```
Program inversare_cuv, nt;
Type reper=^element;
        Element=record
                litera:char;
                next:reper;
            end;
Var v, rf, p : reper;

Begin
        v, rf:=nil;          {stiva vida}
        write(ë cuv, ntul: ë);
        while not eoln do
                begin
                      new(p);
                      read (p^.litera);
                       p^.next:=v, rf;
                       v, rf:=p
                  end;
        if v, rf = nil then writeln (ës-a tastat enter ë)
                        else begin
                                write (ëcuv, ntul inversat este : í);
                                p:=v, rf;
                                repeat
                                        write(p^.litera);
                                        p:=p^.next
                                until p=nil; writeln
                              end
     end.
```

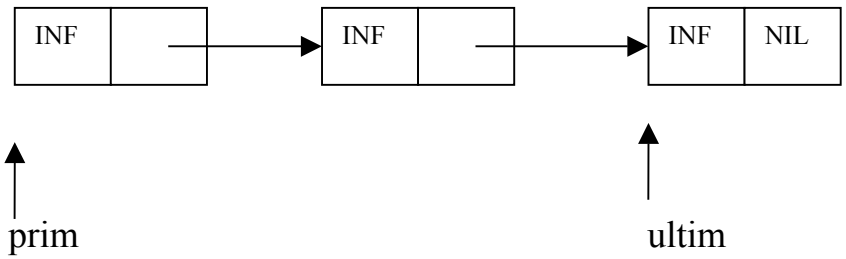
2. COADA(F I F O)

COADA reprezinta o categorie speciala de lista liniara, în care elementele se adauga la un capat (sf, rsit) si se suprima la celalalt (început).

Asemanarea cu o coada care se formeaza la un magazin va usura înelegerea acestui tip de lista. Si denumirea este semnificativa : F I F O = First In ñ First Out ñ adica ì primul venit- primul servitî .

Spre deosebire de stiva , la care operatiile se executa doar la un capat al stivei (v, rf), la coada , adaugarile se fac numai la sf, rsitul cozii (ca si asezarea unei persoane la r, nd), iar iesirile sau stergerile , se realizeaza numai la începutul cozii .

Reprezentarea grafica a unei cozi :



C, nd coada este vida , pointerul prim va indica valoarea NIL. Deci conditia de coada vida este :

prim=nil ;

Variabila reper *prim* are ca valoare adresa primului element din lista, în timp ce variabila reper *ultim,* are ca valoare adresa ultimului element din lista. Deci, spre deosebire de stiva, coada va avea doi pointeri (variabile de tip reper) : *prim* si *ultim*.

Elementul curent a carui adresa este retinuta în variebila reper *p*, contine cele doua zone : -zona de informatie si
-zona de legatura.

Operatiile specifice acestei structuri sunt:

- *crearea listei liniare de tip coada*
- *adaugarea unui element în structura*
- *stergerea unui element din structura*
- *parcurgerea (traversarea) elementelor structurii*

1. Crearea listei de tip coada ñ presupune :

- adaugarea primului element Ó ntr-o coada vida
- adaugarea elementelor la sf, rsitul cozii.

1.1. Adaugarea primului element presupune parcurgerea urmatoarelor etape :

a) Alocarea memoriei pentru primul element:

```
New(prim);
```

b) Initializarea elementului alocat prin precizarea continutului zonei de informatie si al zonei de legatura :

```
prim^.info:= Ö .. sau   Readln( prim^.info)
prim^.next:=nil;
```

c) variabila *ultim* va primi ca valoare adresa primului element (acesta, fiind singurul element din lista, va fi si ultimul) :

```
ultim:=prim;
```

1.2. Adaugarea unui element Ó coada se poate face numai la sf, rsitul cozii, dupa ultimul element si presupune parcurgerea urmatoarelor etape:

a) Alocarea memoriei pentru noul element :

```
New(p)
```

b) initializarea elementului alocat:

```
p^.info:=Ö .   sau      Readln(p^.info)
p^.next:=nil;
```

Adresa de legatura a elementului adaugat este NIL deoarece acesta este ultimul element din lista , dupa el nemaifiind nici un element.

c) legarea elementului Ó structura:

```
ultim^.next:=p;
```

d) variabila *ultim* primeste ca valoare adresa elementului adaugat, care devine ultimul element din lista:

```
ultim:=p;
```

Trebuie sa se tina seama de ordinea în care se executa operatiile, pentru a nu se pierde informatiile. Legarea noului element în structura presupune existenta variabilei dinamice *ultim*^. In cazul în care lista este vida, variabila *ultim* nu exista, de aceea nu este necesara legarea elementului în structura.

2. Parcurgerea elementelor unei cozi: se realizeaza pornind de la primul element al listei (p:= prim) si trec,nd pe r,nd prin fiecare element (p:=p^.next) p,na se ajunge la sf,rsitul listei . În pseudocod parcurgerea se realizeaza astfel:

```
p:= prim
c,t_timp p<>nil executa
        *se parcurge elementul p^
        p:=p^.next
sf,rsit_c,t_timp
```

3. Stergerea unui element al cozii: presupune stergerea primului element al listei .Se parcurg urmatoarele etape:

1. *se salveaza adresa elementului ce va fi sters :*

p:=prim;

2. *se elimina din structura primul element* prin memorarea în variabila reper prim a adresei elementului care urmeaza lui prim, acesta devenind noul prim element:

prim:=prim^.next

3. *se elibereaza memoria ocupata de fostul prim element al cozii :*

Dispose(p)

În cazul în care se sterge unicul element al listei, coada devine vida, iar variabila prim va avea valoarea nil.Procedurile corespunzatoare acestor operatii, consider,nd o lista de tip coada declarata astfel:

```
Type reper=^element;
        element=record
                info:string[10];
                next:reper
          end;
Var prim, ultim,p :reper;
        text:string[10];
```

vor fi urmatoarele:

Crearea cozii

```
Procedure creare;
Begin
        Write(ë dati informatia : ë); readln(text);
        If prim=nil        {coada este vida}
                then  begin                {adaugarea primului element}
                        new(prim);
                        prim^.info:=text;
                        prim^.next:=nil;
                        ultim:=prim;
                      end
                else                       {adaugare element la sf, rsitul cozii}
                        begin
                        new(p);
                        p^.info:=text;
                        p^.next:=nil;
                        ultim^.next:=p
                        ultim:=p
                        end
  end;
```

Parcurgerea cozii:

```
        Procedure listare;
        begin
                if prim=nil        {coada vida}
                        then  writeln (ëcoada este vidaí)
                        else begin
                                p:=prim;
                                repeat
                                        writeln(p^.info);
                                        p:=p^.next
                                until p=nil
                          end
        end;
```

Stergerea unui element din coada

```
Procedure stergere;
 begin
        if prim=nil then writeln(ë lista este vidaí )
                    else begin
                              p:=prim;
                              prim:=prim^.next;
                              dispose(p)
                              end
        end;
```

Un exemplu clasic pentru probleme ce se rezolva utiliz, nd liste de tip coada este problema urmatoare (a carei rezolvare se gaseste ón capitolul cu aplicatii):

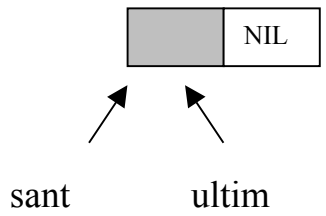
Dispecerizare locomotive

Se considera un depou de locomotive cu o singura linie de cale ferata cu o intrare si o iesire. Sa se scrie programul care realizeaza dispecerizarea locomotivelor din depou, prelucr, nd comenzi de forma: I- intrarea unei locomotive, E- iesirea unei locomotive, L-listarea locomotivelor din depou si S-pentru oprirea programului (si listarea locomotivelor existente ón depou). Locomotivele se identifica printr-un cod.

COADA CU SANTINELA

Pentru eliminarea testului de coada vida la adaugarea unui element , coada poate fi creata cu o santinela ca prim element al cozii;

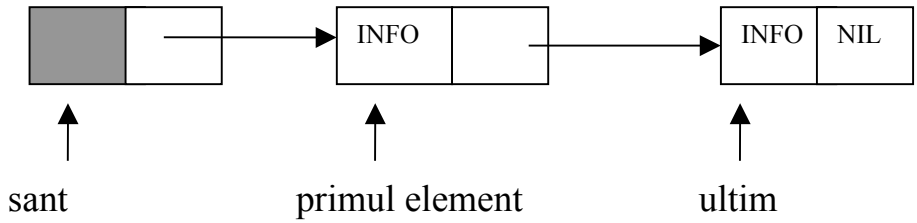
Santinela este o variabila dinamica a carei zona de informatie este neinitializata si care a fost introdusa ca prim element al cozii ón vederea optimizarii operatiilor asupra cozii. O coada vida va cuprinde numai santinela:




```
New(sant);
ultim:=sant;
ultim^.next:=nil;
```

Pentru a se ónelege mai bine rolul santinelei, se va da ca exemplu o coada formata la un magazin la care , pentru bunul mers al lucrurilor , se apeleaza la un om de ordine care supravegheaza modul de servire a consumatorilor, fara a fi considerat consumator (fara a face parte din lista).

Reprezentarea grafica a unei cozi cu santinela :



Operatiile corespunzatoare unei cozi cu santinela sunt:

1. ***Adaugarea unui element Ó coada***-nu mai are importanta daca lista de tip coada este sau nu vida. Adaugarea presupune:

   1.1 *alocarea memoriei pentru noul element:*
   New(p)
   1.2. *initializarea elementului alocat*
   p^.info:=Ö . sau readln(p^.info);
   p^.next:=nil;
   1.3 *legarea elementului Ó structura*
   ultim^.next:=p;
   1.4 *initializarea variabilei ultim cu adresa ultimului element adaugat:*
   ultim:=p

2. ***Parcurgerea elementelor unei cozi cu santinela-*** presupune parcurgerea elementelor cozii, nu si santinela:

```
        p:= sant^. next
        c, t_timp p<>nil executa
                * se parcurge elementul p^
                p:= p^.next
sf, rsit_c, t_timp
```

3.***Stergerea primului element*** al unei cozi cu santinela:

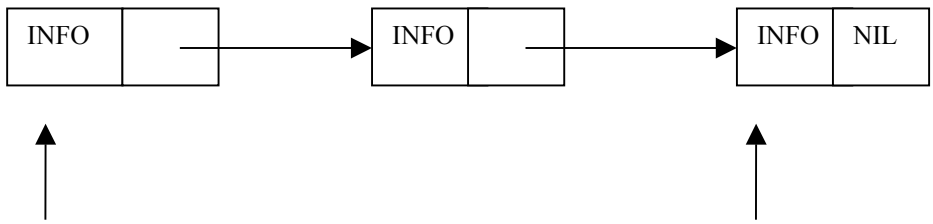
   3.1 *se salveaza adresa santinelei:*
   p:=sant;
   3.2 *se muta santinela pe elementul care ar trebui sters*
   sant:=sant^.next;
   3.3 *se elibereaza memoria ocupata de fosta santinela*
   Dispose(p);

LISTA LINIARA SIMPLU ÎNLANTUITA

Lista linira simplu înlantuita este o lista ale carei elemente pot fi adaugate oriunde în cadrul listei si pot fi sterse , indiferent unde s-ar afla ele. Fiecare element al listei contine o parte cu informatia propriu-zisa si o parte cu adresa urmatorului element din lista. Deci parcurgerea listei se va realiza doar într-un sens si anume de la primul element, a carui adresa se gaseste în variabila reper *prim* spre ultimil element, a carui adresa este memorata în variabila reper *ultim.* Se va nota cu *p* adresa elementului curent al listei.

Ca exemplu de lista liniara se poate da lista copiilor care vor merge într-o excursie, fiecare copil stiind numele (adresa de legatura) copilului care îi urmeaza în lista.

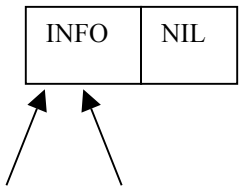
Reprezentarea grafica a unei liste liniare simplu înlantuita :



prim ultim

Lista vida presupune îndeplinirea conditiei prim=nil

Lista cu un singur element poate fi reprezentata astfel:



prim ultim

Operatii cu liste liniare simplu înlantuite:

1. Crearea listei presupune doua etape:

1. *adaugarea primului element*
2. *adaugarea celorlalte elemente*

Un element se poate adauga :

- înaintea elementului p^ , caz în care pot sa apara doua situatii:
  - p=prim
  - p<>prim
- dupa elementul p^, caz în care pot sa apara doua situatii:
  - p=ultim
  - p<>ultim

1.1. adaugarea primului element presupune parcurgerea urmatoarelor etape:

- *alocarea memoriei pentru noul element:*
  new(prim);
- *initializarea noului element:*
  prim^.info:= Ö sau readln(prim^.info)
  prim^.next:=nil;
- *variabila* ultim *primeste ca valoare adresa elementului :*
  ultim:=prim;

1.2. adaugarea unui element :

1.2.1. ***Ó naintea primului element:***

a) se aloca memorie pentru noul element :
new(p);
b) se initializeaza elementul
p^.info:=Ö sau readln(p^.info);
c) se leaga Ó structura elementul adaugat:
p^.next:=prim;
d) variabila prim va primi ca valoare adresa elementului adaugat:
prim:=p;

1.2.2. ***dupa ultimul element:***

a) se aloca memorie pentru noul element:
new(p);
b) se initializeaza elementul:
p^.info:=Ö sau readln(p^.info);
p^.next:=nil;
c) se leaga elementul Ó structura:
ultim^.next:=p;
d) elementul introdus devine ultimul element din lista
ultim:=p;

1.2.3. ***dupa elementul* p^:**

a) se aloca memorie pentru noul element:
new(q);
b) se initializeaza noul element:
q^.info:=Ö sau readln(q^.info);
q^.next:=p^.next;
c) legarea noului element Ó structura:
p^.next:=q;

Ordinea celor doua operatii de legare în structura este esentiala; daca se executa întâi *p^.next:=q* , se pierde adresa elementului care urma dupa p^ înainte de adaugarea lui q^.

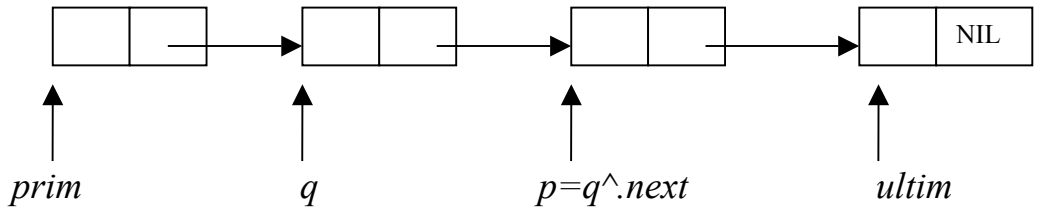
1.2.4. ***Înaintea elementului p^:***

Fiecare element al listei contine adresa elementului urmator în lista; prin urmare, daca am ajuns la elementul p^, nu mai stim adresa elementului precedent în lista, adresa de care avem nevoie pentru a adauga un element înaintea elementului p^.

Avem doua solutii:

a) se pastreaza adresa elementului precedent lui p^, notând acest element cu q^ si folosim relatia :

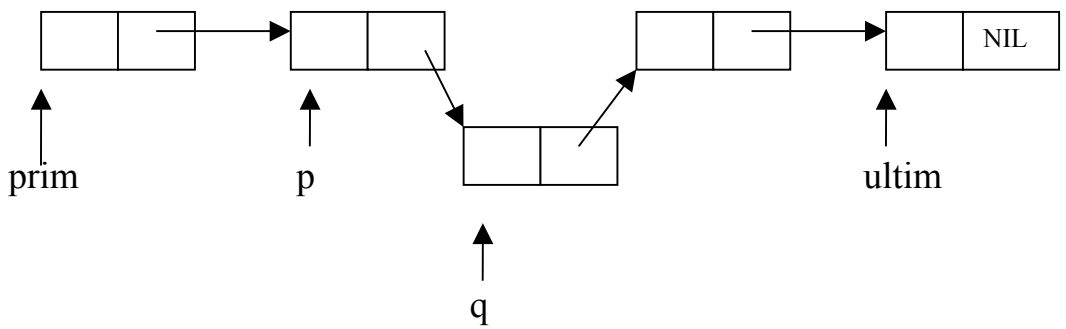
p:=q^.next;



În acest fel, problema adaugarii înaintea elementului *p^* este echivalenta cu problema adaugarii dupa *q^* .

b) elementul q^ trebuie sa apara în lista înaintea elementului p^ . Acest lucru se poate realiza procedând astfel:

- *se aloca memorie pentru noul element :* new(q);
- *se copiaza p^ în noul element notat cu q^:* q^:=p^;
- *se initializeaza p^.info cu valoarea care trebuie adaugata*
- *se leaga elementul p^ de q^ :* p^.next:=q;
- *daca p este ultimul element :* ultim :=q;

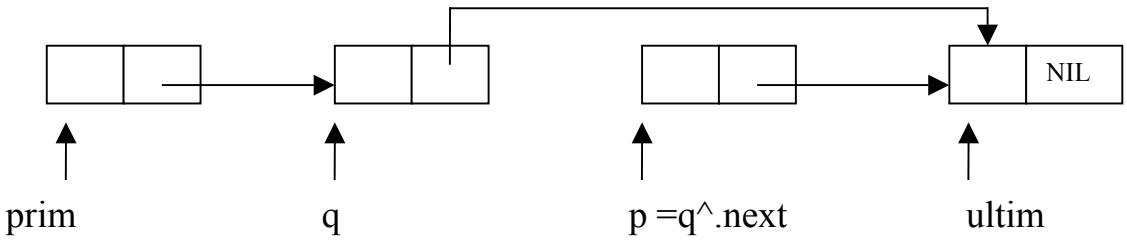


2. ***<u>Stergerea unui element din lista</u>*** ñpot sa apara urmatoarele situatii:

- se sterge primul element din lista
- se sterge ultimul element al listei
- se sterge un element oarecare al listei p^

Stergerea logica a elementului p^ presupune eliminarea lui din structura prin legarea elementului precedent lui p^ cu urmatorul sau element. *Stergerea fizica* se realizeaza cu procedura Dispose. Apare necesitatea cunoasterii adresei elementului precedent. Avem doua solutii:

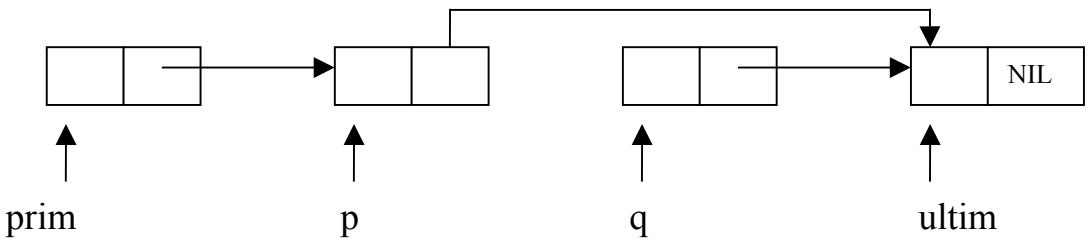
a) pastram adresa elementului precedent (q^) si transformam problema stergerii elementului p^ ón stergerea elementului q^.next^.



Se parcurg urmatoarele etape:

- *se pastreaza adresa elementului precedent* q^;
- *se salveaza adresa elementului care trebuie sters* p:=q^.next;
- *se elimina din structura* q^.next , *leg, nd* q^ *de* p^.next^*:*
  q^.next:=q^.next^.next
- *se elibereaza memoria ocupata de elementul sters :* dispose(p)

b) mutam ón locul elementului care trebuie sters (p^), elementul urmator (p^.next^.) si eliberam memoria ocupata anterior de acesta.



3. Parcurgerea listei (afisarea elementelor listei)-se realizeaza de la primul spre ultimul element al listei , afis, nd informatia continuta de fiecare element, ón cazul ón care se doreste afisarea elementelor listei.
Œn pseudocod vom avea:

```
p:= prim
c, t_timp p<>nil executa
        * se parcurge elementul p^
        p:=p^.next
sf, rsit_c, t_timp
```

Consider, ndu-se data urmatoarea declarare a unei liste simplu ólantuite:

```
Type reper=^element;
        element=record
           info:string[10];
           next:reper
        end;
var prim, ultim,p: reper;
        text:string[10];
```

se vor putea scrie urmatoarele proceduri pentru :

Crearea listei :

```
procedure creare;
  begin
    prim:=nil;
    write('introduceti textul :');readln(text);
    while text<>'' do
      begin  if prim=nil
                  then begin
                          new(prim);
                          prim^.info:=text;
                          prim^.next:=nil;
                          ultim:=prim
                       end
                else  begin
                          new(p);
                          p^.info:=text;
                          p^.next:=nil;
                          ultim^.next:=p;
                          ultim:=p;
                       end;
             write('introduceti textul :');readln(text);
        end;
    ultim^.next:=nil;
end;
```

Adaugarea unui element Óaintea primului element:

```
procedure adaugare_in;  { adaugare la inceputul listei}
begin
    new(p);
    write('introduceti informatia ce se va adauga la inceput: ');
                readln(text);
                p^.info:=text;
                p^.next:=prim;
                prim:=p;
            end;
```

Adaugarea unui element dupa ultimul element:

```
procedure adaugare_sf;  {adaugare la sfarsitul listei}
            begin
                new(p);
                write('introduceti informatia ce se adauga la sfarsit: ');
                readln(text);
                p^.info:=text;
                p^.next:=nil;
    ultim^.next:=p;
                ultim:=p;
            end;
```

Parcurgerea elementelor unei liste:

```
procedure afisare;
        begin
          p:=prim;
          while p<>nil do
            begin
                write(p^.info,'  ');
                p:=p^.next;
            end  ;
          writeln;
        end;
```

Stergerea unui element din lista:

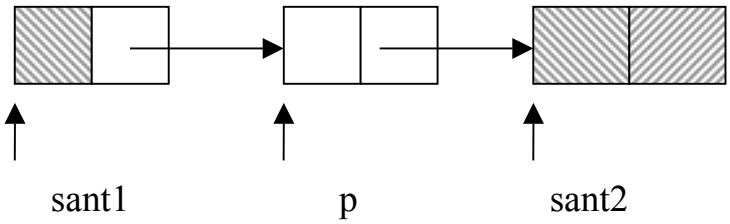
```
procedure sterge;
var q:reper;
  begin
    p:=prim;
    write('introduceti informatia ce se va sterge: ');readln(text);
    if p^.info<>text then
        begin
                q:=p^.next;
                while q^.info<>text do
                                begin
                                    p:=q;
                                    q:=q^.next;
                                end;
                p^.next:=q^.next;
                if q=ultim then ultim:=p;
                dispose(q);
        end
                else
         begin
                prim:=p^.next;
                dispose(p);
        end;
end;
```

Pentru a înlatura dezavantajul creat de numarul mare de cazuri care pot sa apara la tratarea listelor liniare simplu înlantuite se poate crea lista liniara cu santinele.

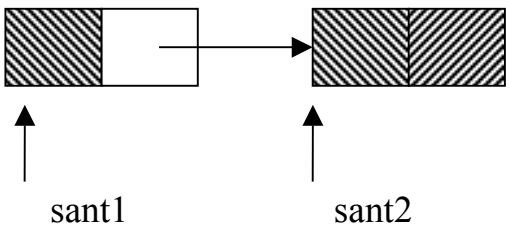
Existenta santinelelor elimina cazurile de lista vida, de adaugare în fata primului element sau dupa ultimul.

<u>Lista oarecare cu santinele</u> - este o lista liniara oarecare în care nu mai avem variabilele *prim* si *ultim* , în care erau retinute adresele primului, respectiv a ultimului element din lista, locul lor fiind luat de doua variabile de tip reper : *sant1* si *sant2,* elemente care nu sunt considerate ca facând parte dintre elementele listei. Rolul santinelei a fost explicat la tratarea cozii cu santinela.

Reprezentarea grafica a unei liste cu santinele:



Lista vida se reprezinta ca având doar cele doua santinele:




```
new(sant1);
new(sant2);
sant1^.next:=sant2
```

Procedurile sunt asemanatoare celor de la lista liniara simplu înlantuita cu deosebirea ca, în locul variabilei *prim* avem variabila *sant1,* iar în locul variabilei *ultim* vom avea variabila *sant2.* Adaugarea înaintea primului element se transforma în adaugarea dupa sant1, iar adaugarea dupa ultimul element se reduce la adaugarea înaintea lui sant2. Conditia de lista vida devine inutila. O particularitate apare la parcurgerea listei , când se vor parcurge numai elementele listei, nu si santinelele:

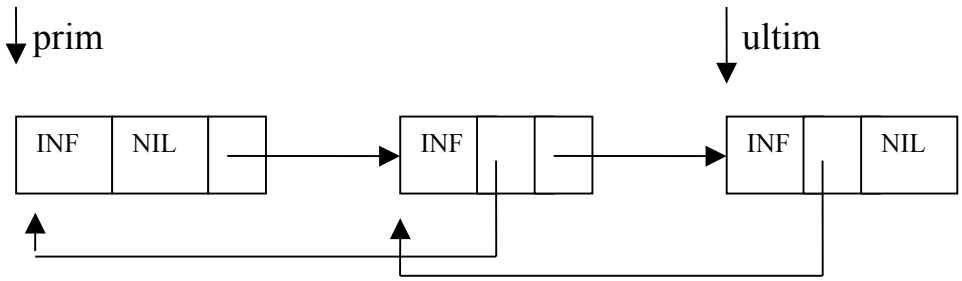
```
p:=sant1;
while p<>sant2 do
        begin
                write(p^.info);
                p:=p^.next
        end;
```

Pentru exemplificare se poate rezolva problema stergerii elementului minim dintr-o lista de numere întregi distincte, folosind o lista liniara creata cu doua santinele: *sant1* si *sant2.*

4. LISTA DUBLU ÎNLANTUITA

Lista dublu înlantuita este un tip special de lista în care informatia de legatura a fiecarui element cuprinde atât adresa elementului precedent în lista, cât si adresa elementului urmator.

Reprezentarea grafica a unei liste dublu înlantuite:

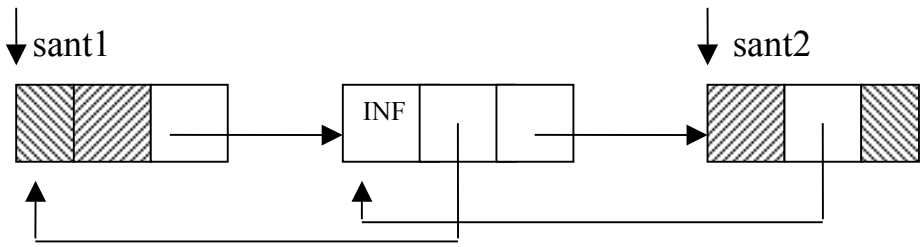


O componenta a listei va contine trei locatii, sau câmpuri:

locatia *info* contine informatia;
locatia *pred* contine adresa componentei anterioare;
locatia *next* contine adresa componentei urmatoare.

Variabilele *prim* si *ultim* vor memora adresa primului, respectiv a ultimului element din lista, iar variabila *p* retine adresa elementului curent.

Ca si în cazul listelor simplu înlantuite , si listele dublu înlantuite se pot crea cu doua santinele. Reprezentarea grafica a unei astfel de liste poate arata astfel:



În cazul listelor dublu înlantuite cu santinele, în locul variabilelor *prim* si *ultim* vom folosi variabilele *sant1,* respectiv *sant2.* Semnificatia notiunii de santinela se cunoaste de la listele anterioare.

Operatiile cu listele dublu Ólantuite sunt:

- ***crearea listei****;*
- ***adaugarea unui element Ól lista****:*
  - *adaugare Óaintea primului element;*
  - *adaugare dupa ultimul element;*
  - *adaugare Óaintea unui element oarecare* p^;
  - *adaugare dupa un element oarecare* p^;
- ***stergerea unui element din lista:***
  - *stergere la st, nga lui p^;*
  - *stergere la dreapta lui p^;*
  - *stergerea primului element al listei;*
  - *stergerea ultimului element al listei;*
- ***parcurgerea elementelor listei;***

Daca lista este creata cu doua santinele , atunci toate elementele listei vor avea un element precedent si un succesor, deci numarul de cazuri , at, t la adaugarea unui element , c, t si la stergerea unui element din lista , se reduce.

De asemenea , existenta celor doua adrese de legatura Ól fiecare element al listei dublu Ólantuite simplifica mult at, t operatiile de adaugare c, t si pe cele de stergere , nemaifiind o problema cunoasterea adresei anterioare elementului curent p^.

Presupun, nd declarata urmatoarea lista:

```
Type reper=^element;
   element=record
     info:string[10];
     pred:reper;
     next:reper
     end;
  var prim,ultim,p:reper;
   text:string[10];
```

putem scrie urmatoarele proceduri:

<u>1. Crearea listei dublu Ólantuita</u> presupune scrierea Ól lista a primului element si adaugarea celorlalte elemente ale listei. Adaugarea primului element presupune:

- *alocarea de spatiu pentru noul element;*
- *memorarea informatiei utile si a informatiei de legatura pentru noul element*
- *variabilele* prim *si* ultim *vor primi ca valoare adresa elementului introdus Ól lista*

```
procedure creare;
  begin
      write(ë informatia: ë);readln(text);
      new(prim);
      prim^.info:=text;
      prim^.pred:=nil;
      prim^.next:=nil;
      ultim:=prim
   end;
```

Prin preocedura de mai sus s-a memorat Ó lista primul element , element a carui Ó scriere se realizeaza Ó tr-un mod diferit de a celorlalte elemente.

2. ***<u>Adaugarea unui element Ó lista dublu Ó lantuita</u>*** ***ñ*** pot sa apara urmatoarele situatii:

Ø **adaugarea unui element la dreapta lui p^ ,** unde **p^** poate fi initial chiar primul element . Se parcurg urmatorii pasi:

- ü *se aloca memorie pentru noul element:* new(p);
- ü *se completeaza c, mpurile de informatie si de adrese*
- ü *se leaga Ó structura noul element:* ultim^next:=p
- ü *elementul nou introdus devine ultimul element din lista* ultim:=p

```
procedure adaugare_dr ( p:reper );
 var nr,i:integer;
 begin
   write('nr. inf:');readln(nr);
    for i:=1 to nr do       {nr reprezinta nr informatiilor introduse}
      begin
         write('adaug informatia: ');readln(text);
         new(p);
         p^.info:=text;
         p^.pred:=ultim;
         p^.next:=nil;
         ultim^.next:=p;
         ultim:=p
       end;
    end;
```

- **adaugarea unui element la st, nga lui p^-**se parcurg aceleasi etape ca la adaugarea la dreapta . Faptul ca avem adresa elementului anterior face ca adaugarea la st, nga sa fie asemanatoare adaugarii la dreapta lui p^.

```
procedure adaug_st(p:reper);
   var  q:reper;
    begin
          new(q);
          write('adaug informatia :');readln(text);
          q^.info:=text;
          q^.pred:=p^.pred;
          q^.next:=p;
          p^.pred^.next:=q;
          p^.pred:=q;
     end;
```

- **adaugarea unui element la Óinceputul listei ñ**presupune legarea noului element Óinaintea lui *prim.*

```
procedure adaug_in;
    begin
          new(p);
          write(ëntroduceti informatia : ë);readln(text);
          p^.info:=text;
          p^.next:=prim;
          p^.pred:=nil;
          prim^.pred:=p;
prim:=p;  end;
```

- **adaugarea la sf, rsitul listei ñ**presupune adaugarea dupa ultimul element,caz asemanator cu adaugarea la dreapta lui p^, atunci c, nd p=ultim

3.Parcurgerea unei liste dublu Ónlantuita-se poate realiza Ón ambele sensuri, av, nd adresele de legatura at, t spre st, nga c, t si spre dreapta. Lista dublu Ónlantuita este o lista simetrica.

a) Parcurgerea de la st, nga la dreapta:

```
procedure parcurg_st;
begin
    p:=prim;
    while p<>nil do
      begin
          writeln(p^.info);
           p:=p^.next
       end;
 end;
```

b) Parcurgerea de la dreapta la st, nga:

```
procedure parcurg_dr;
  begin
       p:=ultim;
       while p<>nil do
              begin
                writeln(p^.info);
                 p:=p^.pred;
              end
   end;
```

4. Stergerea unui element :

- **stergerea elementului oarecare *p^* -**presupune:

  - *Stergerea logica* a elementului se realizeaza prin redirectionarea legaturilor astfel Ónc, t elementul precedent lui p^ - (p^.pred) - va avea ca urmator element elementul urmator lui p^, iar elementul urmator lui p^- (p^.next)- va avea ca element precedent elementul anterior lui p^.

  - *Stergerea fizica* a elementului se realizeaza cu dispose(p);

  Procedura corespunzatoare va fi urmatoarea:

```
procedure sterg;
begin
        write('inf ce va fi stearsa: ');readln(text);
        p:=prim;
        while p^.info<>text do p:=p^.next;
        p^.pred^.next:=p^.next;
        p^.next^.pred:=p^.pred;
        dispose(p)
  end;
```

- **stergerea elementului p^ - c, nd p=prim** *(*stergerea primului element) presupune parcurgerea urmatorilor pasi:

  - se salveaza adresa elementului ce va fi sters *: p:=prim*;
  - variabila prim va lua ca valoare adresa urmatorului element deoarece prin stergerea primului element cel de-al doilea va deveni prim element: *prim:=prim^.next;*
  - noul *prim* element va avea adresa elementului precedent = NIL;
  - se sterge fizic primul element a carui adresa a fost retinuta în p : *dispose*(p).

```
procedure sterg_prim;
begin
        p:=prim;
        prim:=prim^.next;
        prim^.pred:=nil;
        dispose(p);
end;
```

- **-stergerea elementului p^ - c, nd p=ultim** (stergerea ultimului element).

```
procedure sterg_ultim;
begin
        p:= ultim;
        ultim:=ultim^.pred;
        ultim^.next:=nil;
        dispose(p);
end;
```

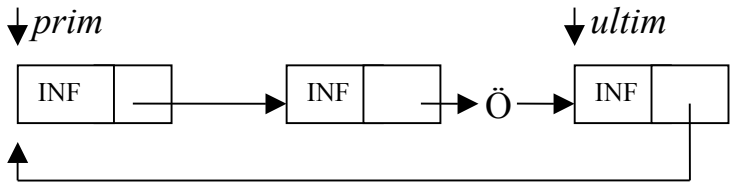
- **stergerea elementului anterior lui p^** - not, nd cu q^ elementul anterior lui p^, si pastr, nd adresa acestuia în q , stergerea elementului anterior lui p^ se face prin stergerea lui q^, a carui adresa se cunoaste.

LISTA CIRCULARA

Lista circulara este o lista cu proprietatea ca elementul urmator ultimului element este primul element al listei. Lista poate fi simplu sau dublu Ólantuita, dupa cum este lista initiala, adica lista la care se leaga dupa ultimul element chiar primul element al listei. Dupa legarea elementelor listei nu mai stim care este primul si ultimul element al listei ,av, nd nevoie de un singur pointer pentru accesarea unui element al listei .

Se pot da exemple de astfel de liste pentru o mai buna Ótelegere a acestei structuri. Un astfel de exemplu este asezarea unor copii Ón cerc si numararea lor prin eliminare p, na la ram, nerea unui singur copil , acesta fiind c, stigatorul jocului. Un astfel de exemplu este ì Jocul lui Josephî . Algoritmul care rezolva aceasta problema este prezentat Ón partea de aplicatii.

Reprezentarea grafica a unei liste circulare simplu Ólantuite :



La listele circulare procedurile vor fi asemanatoare cu cele de la listele din care sunt create aceste liste, tin, nd cont de faptul ca ultimului element al listei Ó urmeaza primul ei element. Astfel o lista circulara dublu Ólantuita va fi si ea simetrica, put, nd fi parcursa Ón ambele sensuri.

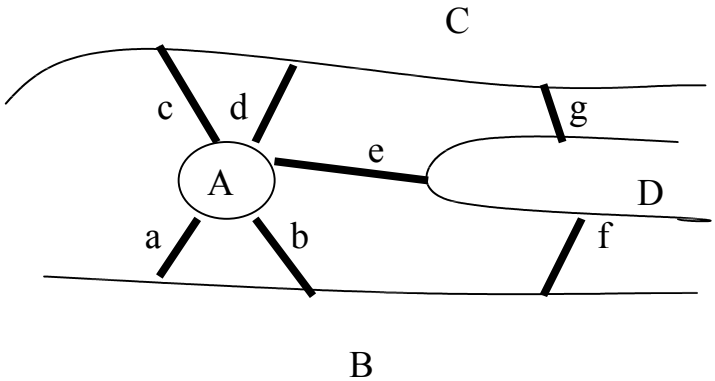
CAP . II

GRAFURI NEORIENTATE

1.GENERALITATI

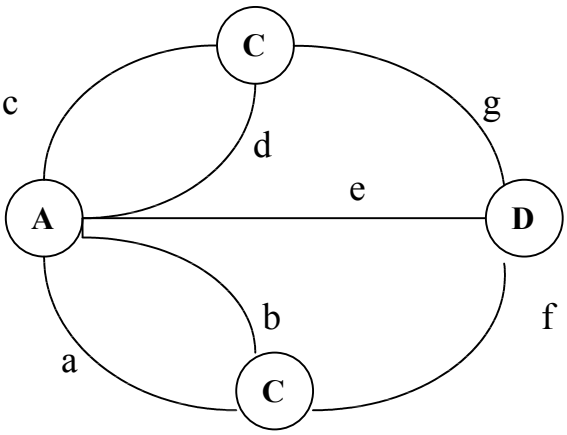
Grafurile se studiaza în clasa a X-a la obiectele: ì Bazele informaticiiî si la ì Aplicatii practice de laboratorî .

Prima lucrare care abordeaza notiunea de graf dateaza din 1736 si se datoreaza matematicianului elvetian Leonard Euler care a folosit un graf pentru a solutiona problema podurilor din Koenigsberg. R,ul Pregat, care trece prin acest oras, îl împarte astfel



Portiunile de uscat A, B, C, D sunt unite între ele prin sapte poduri: a, b, c, d, e, f, g. Problema consta în a determina daca este posibil ca , plec,nd dintr-un punct de pe uscat sa se poata trece pe toate podurile, în final revenindu-se în punctul initial.

Euler a rezolvat aceasta problema introduc,nd un obiect matematic pe care l-a numit graf si care are urmatoarea forma:



Teoria grafurilor are numeroase aplicatii în fizica, chimie, inginerie, economie, sociologie etc.

Aplicatii specifice grafurilor neorientate sunt cele în care se cere :

-sa se determine lungimea minima a lanturilor dintre x si y

-sa se determine un lant de lungime minima de la x la y

-sa se afiseze(daca exista) un ciclu eulerian al unui graf neorientat G cu n v,rfuri si m muchii

-problema existentei Ó tr-un graf oarecare a unui ciclu hamiltonian etc.

Definitie Se numeste **graf neorientat** o pereche ordonata de multimi (X, U), X fiind o multime finita si nevida de elemente numite noduri sau v, rfuri, iar U o multime de perechi neordonate din X, numite muchii.

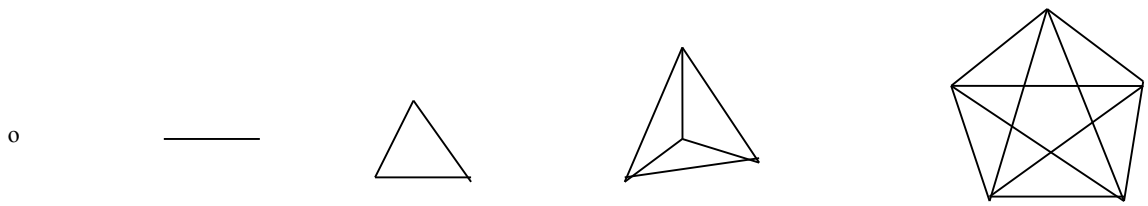
Graful se noteaza cu G= (X , U)

Daca *x* si *y* sunt extremitatile muchiei *u*, spunem ca *u* si *x* sunt *incidente*, la fel ca si *u* si *y* , iar v, rfurile *x* si *y* sunt *adiacente* Ó n G. Deci doua muchii care au o extremitate comuna sunt incidente.

Definitie Un **graf partial** al grafului G= (X , U) este un graf G1= (X , U) a.Ó V $\overline{U}$, adicaG1 are aceeasi multime de v, rfuri ca G, iar multimea de muchii V este , fie chiar U, fie o submultime a acesteia.
Se spune ca un graf partial este indus de multimea V de muchii. Un graf partial al unui graf dat se obtine pastr, nd aceleasi v, rfuri si elimin, nd o parte din muchii.

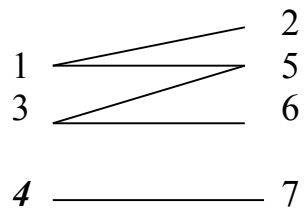
Definitie Un **subgraf** al unui graf G= (X,U) este un graf H=(Y,V) a.Ó Y_X, iar V contine toate muchiile din U care au ambele extremitati Ó n Y.
Vom spune ca subgraful H este indus sau generat de multimea de v, rfuri Y. Un subgraf se obtine elimin, nd o parte din v, rfuri si toate muchiile incidente cu acestea.

Definitie Se numeste **graf complet** cu *n* v, rfuri un graf care are proprietatea ca orice doua noduri diferite sunt adiacente.
Propozitie Un graf complet K_n are $\frac{n(n-1)}{2}$ muchii.
Primele 5 grafuri complete sunt:



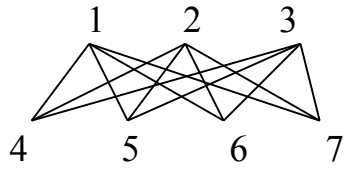
K_1 K_2 K_3 K_4 K_5

Definitie Un graf G=(X,U) se numeste **graf bipartit** daca exista doua multimi nevide A si B a.Ó X=A 4 B , A 3 B=‘ si orice muchie *u* a lui G are o extremitate Ó n A si cealalalta Ó n B. Multimile A si B formeaza o partitie a lui X.



A={1,3,4}
B={2,5,6,7}

Definitie Un **graf bipartit** se numeste **complet** daca pentru orice x din A si orice y din B exista în G muchia [x,y].

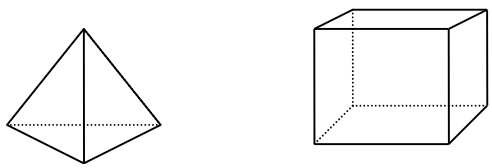


A={1,2,3}
B={4,5,6,7}

Daca A are p elemente , iar B are q elemente, numarul total de muchii ale unui graf bipartit complet este p*q , iar graful se noteaza $K_{p,q}$.

Definitie Se numeste **graf regulat** graful în care toate v, rfurile au grade egale.

Definitie Un **graf K-regulat** este un graf regulat în care gradul comun al v, rfurilor este K.
Exemple: tetraedrul, cubul,octoedrul etc.



Definitie Se numeste **lant** în graful G, succesiunea de v, rfuri $L=[x_1,x_2, \ldots x_k]$ cu proprietatea ca orice doua v, rfuri consecutive din L sunt adiacente, adica $[x_1,x_2]$, $[x_2,x_3], \ldots, [x_{k-1},x_k] \in U$. Varfurile x_1 si x_k se numesc extremitatile lantului , iar numarul de muchii ce apar în L se numeste lungimea lantului.

Definitie Daca v, rfurile $x_1,x_2, \ldots x_k$ sunt diferite doua c, te doua, atunci lantul L se numeste **lant elementar**. În caz contrar lantul este neelementar.
Un lant L poate fi interpretat ca traseul unei deplasari de la x_1 la x_k pe muchiile lantului.

Definitie Se numeste **ciclu** în G un lant L pentru care $x_1=x_k$, si toate muchiile sunt diferite doua c, te doua.

Definitie Se numeste **ciclu elementar** un ciclu care are proprietatea ca oricare doua v, rfuri ale sale , cu exceptia primului si a ultimului, sunt diferite doua c, te doua.

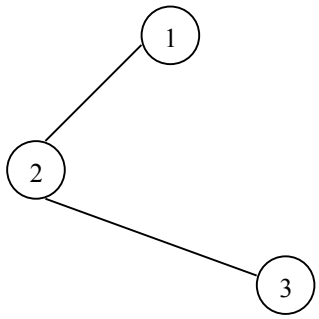
2. REPREZENTAREA GRAFURILOR NEORIENTATE

Exista mai multe modalitati de a reprezenta graful G=(X,U). Aceste reprezentari vor fi utilizate în prelucrarea grafurilor cu ajutorul programelor care implementeaza pe calculator algoritmii respectivi.

- Se precizeaza numarul de vârfuri (n) si matricea de adiacenta A a grafului, care este o matrice patratica de ordinul n , cu elementele:

$$A[i,j]=\begin{cases}1, & \text{daca } [i,j]\in U\\ 0, & \text{în caz contrar}\end{cases}$$

Aceasta matrice este o matrice simetrica. De exemplu, pentru graful urmator vom avea:



$$A=\begin{vmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 0\end{vmatrix}$$

- Se precizeaza nr n de vârfuri si, pentru fiecare vârf i, lista Li a vecinilor sai, adica lista vârfurilor j pentru care [i,j] $\in$ U.

Vârful i	Lista Li a vecinilor sai
1	2
2	1,3
3	2

Acest mod de reprezentare poate utiliza si alocarea dinamica a memoriei, precum si un tablou bidimensional T cu doua linii si n+2*m coloane, unde *m* este nr. de muchii, iar *n* numarul de vârfuri ale lui G.

T [1, i]=i, i=1,n

T [2, i] reprezinta indicele coloanei din T în care este dat primul element din lista Li a vecinilor lui i ; daca I este v,rf izolat, atunci T[2, i]=0; daca T[2, i]=j, atunci T[1,j] este primul dintre vecinii lui i, iar T[2, j] este coloana în T în care apare urmatorul element din lista Li. Daca u este indicele lui T în care apare ultimul element w din Li, atunci T[1,u]=w si T[2,u]=0.

Pentru graful de mai sus vom avea:

i	1	2	3	4	5	6	7
T[1,i]	1	2	3	2	1	3	2
T[2,i]	4	5	7	0	6	0	0

- Se dau numarul n de v,rfuri, numarul m de muchii, precum si doua tablouri unidimensionale e1 si e2 cu c,te m componente fiecare, contin,nd extremitatile muchiilor grafului, adica U={ [e1[1], e2[1]] , [e1[2], e2[2]],Ö .,[e1[m],e2[m]]}.

- O varianta de implementare mai naturala ar fi aceea de a defini un tip de data ëmuchieí utiliz,nd tipul ïregistrare, care sa contina cele doua extremitati ale unei muchii, si apoi de a defini un tablou cu n componente de acest tip:

```
Type muchie=record
        x,y:byte;
        end;
Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö
var u:array[1..n] of muchie;
```

Referirea la extremitatile muchiei se face prin : u[i] .x, respectiv u[i] .y. Acest mod de reprezentare permite ïglobarea natulara în tipul de date *muchie* si a altor informatii asociate muchiilor(cost, lungime) si este utilizata în problemele în care muchiile se prelucreaza succesiv, eventual dupa o modificare a ordinii lor în tabloul u.

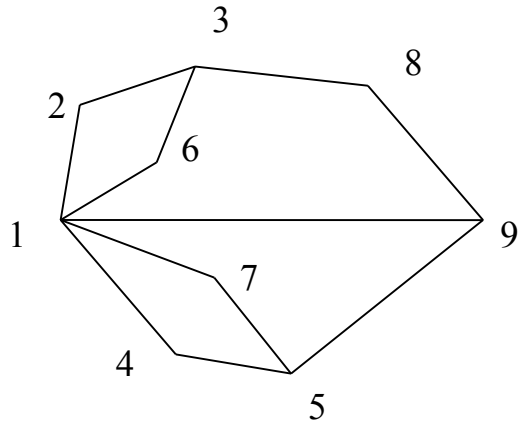
3. GRAFURI EULERIENE

Definitie Se numeste **ciclu eulerian** ´ıntr-un graf G, un ciclu care contine toate muchiile grafului.

Definitie Se numeste **graf eulerian** un graf care contine un ciclu eulerian.

Definitie Se numeste **lant eulerian** un lant care contine toate muchiile grafului, fiecare muchie fiind prezenta o singura data.

Teorema **:** *Un graf G fara v, rfuri izolate este eulerian daca si numai daca este conex si gradele tuturor v, rfurilor sunt numere pare.*
Ex.:



Graf eulerian

Teorema de mai sus da o conditie necesara si suficienta ca un graf care nu contine v, rfuri izolate sa fie eulerian. Nu ne propunem sa demonstram aceasta teorema. Ne intereseaza algoritmul cu ajutorul caruia putem construi un ciclu eulerian ´ıntr-un graf despre care stim ca este eulerian; pentru aceasta se vor parcurge urmatorii pasi:

Pas 1. Plecam dintr-un v, rf oarecare, fie acesta v, rful 1. Construim un prim ciclu C parcurg, nd v, rfuri accesibile din aproape ´ın aproape pe muchii din graf, micsor, nd corespunzator gradele v, rfurilor parcurse.

Pas 2. Alegem pentru continuare(daca aceasta mai este posibila) , un v, rf al ciclului C pentru care mai exista muchii incidente cu el, neluate ´ınca. Construim astfel un nou ciclu C1 pe care ´ıl concatenam cu ciclul C, obtin, nd un ciclu mai lung.

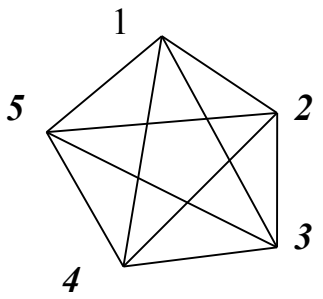
Repetam pasul 2 at, ta timp c, t mai exista muchii care nu au fost ´ınca incluse ´ın ciclul C.

4. GRAFURI HAMILTONIENE

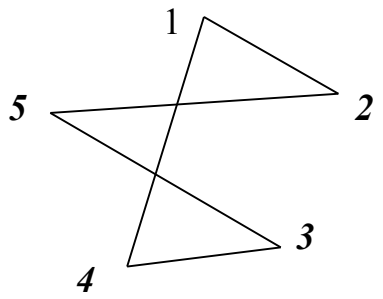
Definitie Se numeste **ciclu hamiltonian** un ciclu elementar care contine toate v, rfurile grafului G.

Definitie Se numeste **graf hamiltonian** un graf care contine un ciclu hamiltonian.

Definitie Se numeste **lant hamiltonian** un lant elementar care contine toate v, rfurile grafului.



Graf hamiltonian



Ciclu hamiltonian

Teorema : *Graful complet K_n este hamiltonian*

Teorema: *Daca G=(X,U) este un graf cu n>=3 v, rfuri, astfel Ó c, t gradul fiecarui v, rf* x χX *satisface conditia:*

$$d(x) \geq \frac{n}{2}$$

atunci G este hamiltonian.

Œ n partea de aplicatii se va prezenta algoritmul care determina toate ciclurile hamiltoniene existente Ó tr-un graf (care se dovedeste a fi hamiltonian).

Acest algoritm , foloseste metoda backtracking. Ciclurile se genereaza utiliz, nd un vector x cu n componente.

Conditiile interne ce trebuiesc satisfacute sunt urmatoarele:

x[i] ! x[j] … (i,j)χ{1,2,Ö ,n}, i!j

[x[1], x[2]] , [x[2], x[3]] ,Ö .,[x[n-1], x[n]] χ U si [x[n], x[1]] χ U

Pentru a nu produce de mai multe ori acelasi ciclu, vom fixa x[1]=1. Oricum fiecare ciclu hamiltonian este generat de doua ori, diferind ordinea de parcurgere a nodurilor. .Putem evita acest lucru memor, nd ciclurile generate si , de fiecare data c, nd terminam generarea unuia, sa verificam daca el a mai fost generat o data.

Pas 1 : sol:=9; *{variabila sol numara ciclurile hamiltoniene}*
x[1]:=1; *{se fixeaza primul v, rf al ciclului}*
k:=2; x[2]:=1; *{se pune ó x[2] valoarea 1 din care se va deduce prima valoare posibila a acestei componente}*

Pas 2 : at, ta timp c, t k>1 executa
v:=0;
at, ta timp c, t (x[k]+1 [n) and (v=0) executa
x[k]:=x[k]+1;
v:=1; *{se porneste de la ideea ca pozitionarea facuta este buna si se modifica v ó caz contrar}*
pentru i:=1,k-1 executa *{v, rfurile trebuie sa fie diferite}*
daca x[i]=x[k] atunci v:=0;
daca a[x[k-1], x[k]]=0 atunci v:=0; *{se testeaza existenta muchiei ótre x[k-1] si x[k]}*
daca v=0 atunci k:=k-1 *{se face un pas ópapoi }*
altfel daca (k=n) and (a[x[n],1]=1)
atunci sol:=sol+1
scrie ciclul ham. obtinut
altfel daca k<n atunci k:=k+1;
x[k]:=1;
{se merge la urmatoarea componenta, care se initializeaza cu valoarea 1, din care se va deduce prima valoare posibila}

O problema órudita cu cea a gasirii ótr-un graf a unui ciclu hamiltonian este ***problema voiajorului comercial:***

Un voiajor comercial trebuie sprezinte ó n orase produsele fabricii pe care o reprezinta, dupa care se ótoarce ó orasul din care a plecat. Cunosc, ndu-se costul deplasarii ótre oricare doua dintre cele n orase, se cere sa se determine un traseu care sa viziteze o singura data cele n orase si care sa aiba costul total minim.

Cu alte cuvinte, problema cere sa se determine un ciclu hamiltonian de cost minim ó graful K_n ale carui v, rfuri sunt cele n orase, iar costul unui ciclu reprezinta suma costurilor muchiilor sale.

Rezolvarea acestei probleme va fi prezentata ó partea de aplicatii.

5. PARCURGEREA GRAFURILOR NEORIENTATE

Prin parcurgerea unui graf neorientat se Ó ntelege examinarea Ó n mod sistematic a nodurilor sale, plec, nd dintr-un v, rf dat ***i*** a.Ó fiecare nod accesibil din ***i*** pe muchii adiacente doua c, te doua, sa fie atins o singura data.

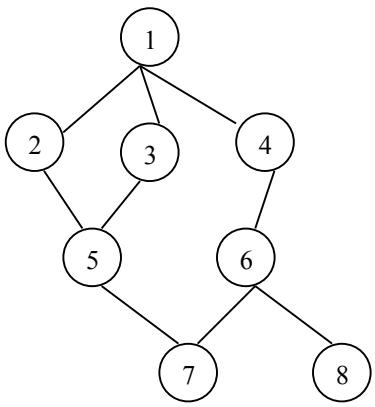
Metode de parcurgere a grafurilor neorientate :

- ***metoda de parcurgere DF*** (Dept First) sau parcurgerea Ó n ad, ncime, poate fi aplicata at, t Ó n cazul grafurilor neorientate c, t si Ó n cel al grafurilor orientate.

Parcurgerea Ó ncepe cu un v, rf initial dat i si continua cu primul dintre vecinii sai nevizitati , fie acesta j . Œ n continuare se procedeaza similar cu v, rful j, trec, ndu-se la primul dintre vecinii sai, nevizitati Ó nca.

De exemplu, pentru graful de mai jos, Ó n cazul Ó n care se Ó ncepe cu nodul 1 si alegerea nodurilor se face Ó n ordine crescatoare,

parcurgerea DF este :1,2,5,3,7,6,4,8 .



Pentru implementarea algoritmului se utilizeaza vectorul *viz* si o stiva *S* care ne permite sa plecam Ó n fiecare moment de la v, rful curent spre primul dintre vecinii sai nevizitati, acesta din urma fiind plasat Ó n v, rful stivei. Œ n vectorul *urm* vom determina Ó n fiecare moment urmatorul nod ce va fi vizitat dupa nodul *j* (atunci c, nd acesta exista).

Notam cu *ps* pointerul de stiva. Se parcurge linia *j* din A Ó ncep, nd cu urmatorul element, p, na este gasit un vecin al lui j nevizitat Ó nca. Daca el este gasit, este plasat Ó n v, rful stivei , marind si *ps* , daca nu, *ps* se micsoreaza cu 1, Ó ncerc, nd sa continuam cu urmatorul element din stiva.

Algoritmul în pseudocod este urmatorul:

Pas 1. Pentru j:=1,n executa viz[j]:=0; *{toate v, rfurile sunt nevizitate}*
urm[j]:=0; *{pornind din 0, vom determina prima valoare posibila pentru primul nod spre care se poate pleca din v, rful j}*
sf, rsit pentru

Pas 2. S[1]:=i; *{plasam în v, rful stivei S v, rful i }*
ps:=1; *{pointerul de stiva* ps *pointeaza spre v, rful stivei }*

Pas 3. At, ta timp c, t ps=1 executa *{stiva este nevida}*
j:=S[ps]; *{ scoate nodul din stiva}*
determina primul dintre vecinii k ai lui j, nevizitati în ca
urm[j]:=k; *{pune k în urm[j]}*
daca nu exista un astfel de k **atunci** ps:=ps-1
{se coboara în stiva}
altfel scrie k; *{ v, rful k este vizitat}*
viz[k]:=1;*{se mareste ps pentru a-l*
S[ps]:=k; *memora pe k în S}*

Pas 4 . stop

Programul Pascal care implementeaza algoritmul va fi prezentat în partea de aplicatii.

- ***metoda de parcurgere BF*** (Breath-First) ñ sau parcurgerea în latime.

Parcurgerea în latime presupune vizitarea în t, i a v, rfului initial i, apoi a vecinilor acestuia, apoi a vecinilor nevizitati ai acestora s.a.m.d.

Pentru graful de mai sus , parcurgerea BF corespunzatoare va fi: 1,2,3,4,5.6.7, 8 .

Pentru constructia practica a algoritmului, în vederea alegerii la un moment dat, dintre toti vecinii unui v, rf , pe acela nevizitat în ca si care îndeplineste conditia impusa, vom folosi un vector *viz* cu n componente, astfel:

$$\ldots j \chi \{1,2,\ldots,n\}, \quad viz[j]=\begin{cases} 1, & \text{daca v, rful j a fost vizitat} \\ 0, & \text{în caz contrar} \end{cases}$$

Œn vectorul C vom gestiona o coada ón care prelucrarea unui v, rf v aflat la un capat al cozii consta ón introducerea ón celalalt capat al ei a tuturor v, rfurilor j vecine cu v, nevizitate ónca. Evident. Initial v este egal cu i, v, rful dat.

Algoritmul ón pseudocod este urmatorul:

Pas 1. Pentru j:=1,n executa viz[j]:=0;

Pas 2. C[1];=i; *{In coada C memoram initial doar v, rful i;}*
p:=1;u:=1; *{cu p pointam la primul element al cozii iar cu u la ultimul}*
viz[i]:=1*; {se viziteaza v, rful I}*

Pas 3. At, ta timp c, t p[u executa {coada este nevida}

Pas 3. 1. V:=C[p]; *{scoate v, rful urmator din coada}*

Pas 3. 2. Pentru toti vecinii j ai lui v, nevizitati ónca, executa
u:=u+1;
C[u]:=j; {adauga v, rful j ón coada C}
Viziteaza v, rful j;
viz[j]:=1; {v, rful a fost vizitat}

Pas 3. 3. p:=p+1;{se trece la urmatorul element ce va fi scos din coada C}

Pas 4. Stop.

Programul Pascal de parcurgere a grafurilor neorientate prin metoda BF va fi prezentat ón partea de aplicatii.

- ***Metoda de parcurgere D*** (depth) ñón cazul ón care lista folosita este o stiva se obtine parcurgerea depth.

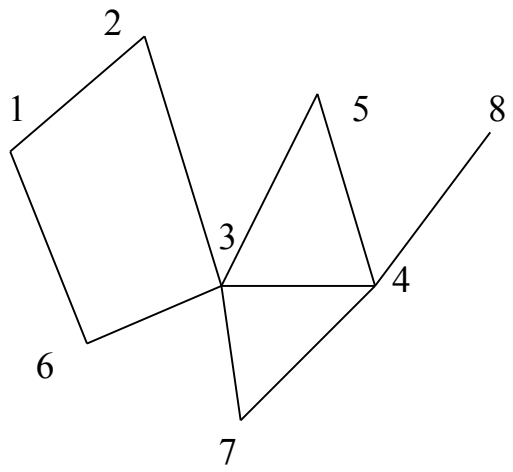
Aceasta se deosebeste de parcurgerea BF prin faptul ca prelucreaza mereuultimul nod la care s-a ajuns. Pentru exemplul considerat, parcurgerea D se va face ón ordinea : 1,2,3,4,5,6,7,8,5.

La fel ca si cazul algoritmului DF, algoritmii BF si D pot fi folositi pentru parcurgerea unui graf neorientat care nu este conex sau pentru grafuri orientate.

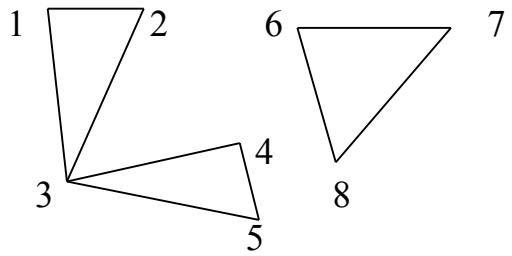
Cele mai utilizate metode de parcurgere sunt DF si BF.

6. CONEXITATE

Definitie Se numeste **graf conex** un graf cu proprietatea ca pentru oricare doua v, rfuri x si y diferite ale sale exista un lant care le leaga.



Graf conex



Graf neconex

Definitie Se numeste **componenta conexa** a grafului G=(X,U), un subgraf C=(X1, U1) conex al lui G care sa lege un v, rf din X1 cu un v, rf din X-X1.

Prezentam algoritmul care verifica daca un graf este conex. Acest algoritm se bazeaza pe parcurgerea BF. Ín cazul ín care graful nu este conex vom cere sa se determine numarul componentelor conexe ale grafului dat.

Se stie ca ín urma parcurgerii BF a unui graf obtinem , o lista a v, rfurilor care, de fapt, reprezinta multimea tuturor v, rfurilor care sunt legate prin lanturi de un v, rf dat.

Algoritmul este urmatorul:

Pas 1: Se alege un v, rf x (ex> x=1). Se initializeaza numarul nc de componente conexe cu 0 (nc:=0_;

Pas 2: Se mareste nc cu 1 (nc:=nc+1). Se determina si se afiseaza multimea v, rfurilor legate de x prin lanturi, utiliz, nd parcurgerea BF , lu, nd x ca v, rf de plecare. Aceasta reprezinta multimea de v, rfuri ale componentei conexe nc . Daca toate v, rfurile au fost vizitate, se trece la pasul 3. Daca nu, se alege un v, rf x ín ca nevizitat si se reia pasul 2.

Pas 3: Se testeaza nc. Daca nc este 1, se concluzioneaza ca graful este conex. Daca nu, se precizeaza numarul total de componente conexe.

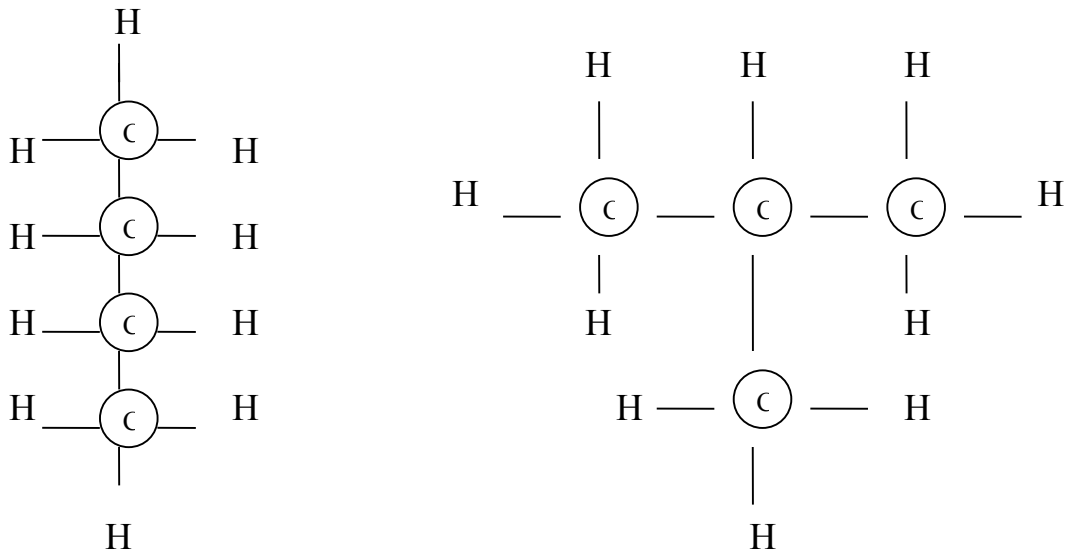
CAP . III

ARBORI

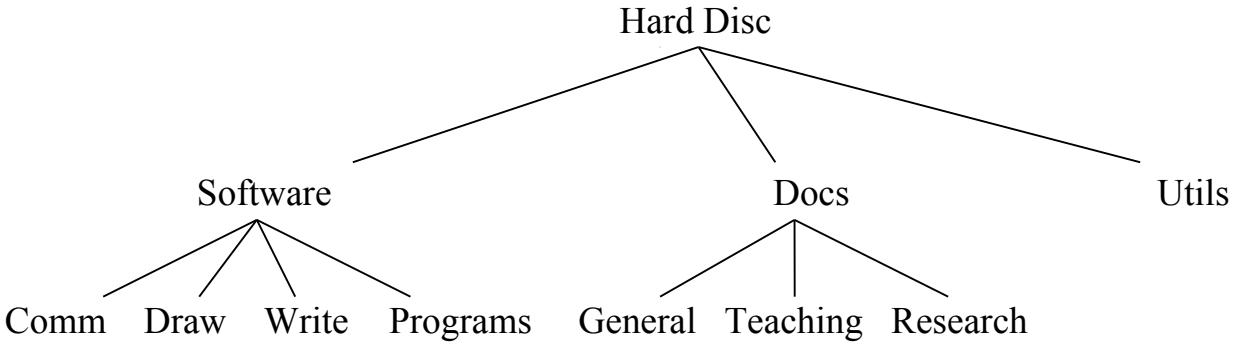
1. GENERALITATI:

Œn clasa grafurilor conexe, arborii reprezinta grafurile cele mai simple ca structura si cele mai frecvent utilizate Ón practica. De studiul lor s-au ocupat matematicieni si fizicieni de seama: Cayley a studiat arborii pentru aplicatiile lor Ón chimia organica, iar Kirchhoff a studiat aceasta categorie de grafuri pornind de la studiul retelelor electrice. Termenul de arbore a fost introdus de Cayley Ón 1857, plec, nd de la o analogie botanica.

Ex. 1. Doua hidrocarburi saturate : butanul si izobutanul , care au aceeasi formula chimica C_4H_{10} pot fi reprezentate astfel:



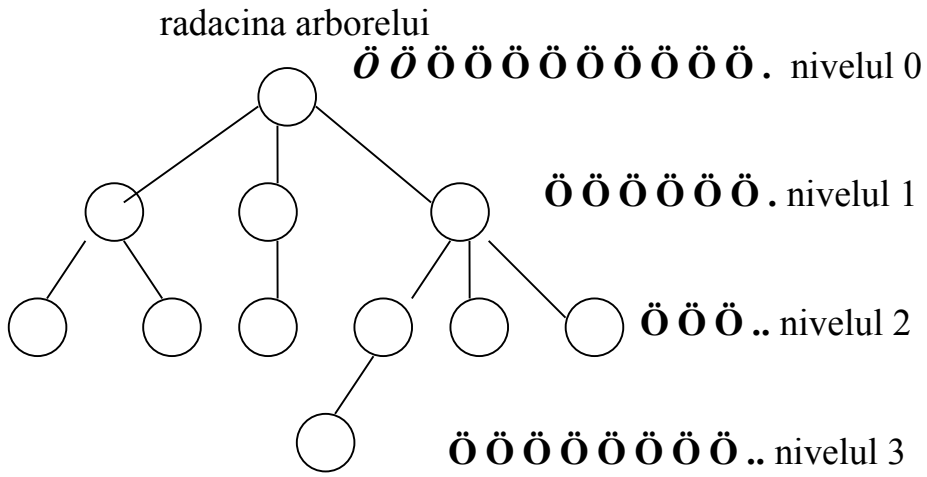
Ex.2. Directoarele sistemului de operare sunt structurate sub forma unui arbore



3. Arborele genealogic al unei persoane este un alt exemplu de structura arborescenta, exemplu care Ón va ajuta pe elevi sa Ónteleaga mai usor acest tip de structura de date.

Œntr-o stuctura de tip arbore, elementele sunt structurate pe niveluri astfel:

- pe primul nivel (nivel 0) exista un element unic numit *radacina* , de care sunt legate elementele de pe nivelul urmator (nivelul 1);
- pe urmatorul nivel se gasesc elementele legate de radacina, elemente care se vor numi *fii* si care formeaza nivelul 1;
- pe celelalte niveluri se gasesc elemente legate de elementele de pe nivelurile anterioare si care , la r, ndul lor pot avea fii. Œn cazul Ó care un element nu mai are nici un fiu, se numeste element *terminal* sau *frunza.*



Definitie : *Se defineste* **<u>un arbore</u>** *ca fiind un graf conex si fara cicluri*.

Se vor reaminti elevilor notiunile de graf conex si de ciclu:

- Un graf conex este un graf Ó care pentru oricare doua v, rfuri exista un lant care le leaga.
- Un ciclu Ó G este un lant pentru care ultimul element al lantului este chiar primul element al lui si toate muchiile sunt diferite doua c, te doua.

Fac, ndu-se analogia cu arborele genealogic se vor defini notiunile cu care se vehiculeaza Ó prelucrarea arborilor :

Fiecare element se numeste *tata* pentru elementele de pe nivelul urmator care sunt legate de el, acestea numindu-se *fiii* elementului considerat. Nodurile care au acelasi tata se numesc *frati*. Fiecare nod are un singur tata , cu exceptia radacinii care nu are tata. Nodurile care nu au descendenti se numesc *frunze* .

Definitie *Se numeste* **<u>arbore partial</u>** *un graf partial H al grafului G care Ó plus este si arbore.*

Corolar: *Un graf G=(X,U) contine un arbore partial daca si numai daca G este conex.*

Propozitia 1. *Orice arbore H=(X,V) cu n ¦ 2 v, rfuri contine cel putin doua v, rfuri terminale.*

Propozitia 2. *Orice arbore cu n v, rfuri are n-1 muchii.*

Definitie *Un graf G=(X,U) care nu contine cicluri se numeste* **graf aciclic.**

Definitie *Un graf G care nu contine cicluri se numeste* **padure.**

APLICATII :

Œn diverse aplicatii (proiectarea unor retele de transport, de comunicatii, de alimentarii cu apa, de energie electrica etc.) apare frecvent problema determinarii ***arborilor partiali de cost minim*** care sa satisfaca restrictii de conexitate si sa minimizeze (maximizeze) anumite lungimi, costuri, preturi .

Se poate cere astfel : sa se determine un arbore partial de cost minim, sa se determine toti arborii partiali minimi sau sa se determine numai aceia care satisfac diverse restrictii de optimizare.

Arborele partial de cost minim este arborele partial la care Œntre oricare doua noduri exista un drum, iar, Œn plus, suma muchiilor este minima. O problema concreta Œn care intervine problema determinarii arborelui partial de cost minim este cea a conectarii oraselor cu cost minim.

Pentru determinarea unui APM al unui graf conex , sunt cunoscute mai multe metode. Vor fi prezentati cei doi algoritmi de determinare a unui APM cunoscuti ca:

- ***Algoritmul lui Kruskal***
- ***Algoritmul lui Prim***

Un mare numar de aplicatii utilizeaza Œn rezolvarea lor ***arborii binari*** . Astfel ei sunt utilizati pentru:

- Memorarea si regasirea rapida a unor informatii (arborele binar de cautare)
- Reprezentarea arborelui genealogic (arbore oarecare) cu ajutorul unui arbore binar;
- Notatia poloneza a expresiilor aritmetice;
- Sortarea elementelor unui sir prin selectie arborescenta (arbori de sortare);
- Reprezentarea functiilor compuse cu ajutorul arborilor binari.
- Arborii binari perfect echilibrati etc.

2. ARBORE PARTIAL DE COST MINIM

Fie un graf G = (X,U) conex, X = {1,2,3,Ö ,n}, si o functie c : UτR, care asociaza fiecarei muchii u, un numar real pozitiv c(u), numit **costul sau.**

Definitie *pentru un graf partial H = (X, V) al lui G, costul sau reprezinta suma costurilor muchiilor sale, adica:*

$$c(H) = \sum_{u \in V} c(u)$$

Pentru un arbore partial de cost minim folosim notatia prescurtata APM.

Problema : *Sa se determine un graf partial H al lui G care sa fie conex si sa aiba costul minim.*

Aceasta problema este cunoscuta si sub numele de ì***problema conectarii cu cost minim a oraselorî ,*** deoarece putem sa interpretam cele n v, rfuri ale grafului ca fiind orase, iar costul muchiei [i , j] ca reprezent, nd costul conectarii directe a oraselor i si j. Un arbore partial conex reprezinta modalitatea ortima din punct de vedere financiar de a lega direct unele perechi de orase a.Ó ó n final orice doua orase sa fie conectate (direct sau prin intermediul altora).

Propozitie: *Pentru graful G conex, cu functia de cost c, exista un graf partial H conex si de cost minim, care, ó plus , este arbore.*

Presupunem cunoscute teoremele care caracterizeaza o padure, respectiv un arbore:

Teorema *(caracterizarea unei paduri). Graful G este o padure daca si numai daca pentru oricare doua noduri ale lui G exista cel mult un lant care le uneste.*

Teorema *(caracterizarea unui arbore). Fie graful G cu n noduri. Urmatoarele conditii sunt echivalente:*

1) *G este un arbore;*
2) *Pentru oricare doua noduri distincte, exista un unic lant care le uneste;*
3) *G este aciclic si are n-1 muchii;*
4) *G este conex si are n-1 muchii;*
5) *G este maximal ó raport cu proprietatea de aciclitate;*
6) *G este minimal ó raport cu proprietatea de conexitate.*

Propozitie *Pentru orice graf conex de tip (n,m) exista relatia m¦ n-1, egalitatea av, nd loc c, nd graful este un arbore.*

Teorema *Orice graf conex contine cel putin un arbore partial.*

2.1. ALGORITMUL LUI KRUSKAL

Av, nd la baza metoda Greedy pentru rezolvarea unei probleme de optimizare, algoritmul construieste un arbore partial optim astfel:

- initial se pleaca de la graful partial vid;
- la fiecare pas se alege acea muchie de cost minim din cele neselectate anterior si care nu formeaza ciclu cu cele selectate deja;
- construirea arborelui partial optim al unui graf conex cu n noduri se sf, rseste atunci c, nd arborele construit are n-1 muchii.

Algoritmul lui Kruskal prezentat ó pseudocod:

citeste *n,m ,(x_i , y_i , c_i , i:= 1,m)*
* se ordoneaza muchiile x_i , y_i crescator dupa costurile c_i
Li r i , i:=1,n { se initializeaza marcajele v, rfurilor*}*
i r 0; costr0;
pentru *j:=1,m* executa
daca *L x_j !Ly_j*
{extremitatile muchiei *j* sunt ó componente conexe diferite}

atunci
iri+1
Si rj { se adauga muchia j ó arbore *}*
cost r cost + c_j
marcaj1 r L x_j { se unifica componentele conexe*}*
marcaj2 r Ly_j
pentru *k=1,n* executa
daca *Lk=marcaj2* atunci *Lkrmarcaj1*

daca *i=n-1*
atunci *scrie* solutia *S* si costul *c*
**oprire fortata*

scrie *ë Graful initial este neconex ë*
stop.

Programul Pascal va fi prezentat ó partea de aplicatii.

2.2. ALGORITMUL LUI PRIM

Œn timp ce algoritmul lui Kruskal consruieste un APM prin construirea multimii muchiilor arborelui minimal , adaug, ndu-i la fiecare pas muchia cu cel mai mic cost care nu formeaza cicluri cu muchiile aflate ón matricea muchiilor arborelui minimal , algoritmul lui Prim construieste un arbore partial optim minim pornind cu o multime formata dintr-un singur nod si o multime de muchii, vida. Arborele se construieste treptat prin adaugarea la fiecare pas a muchiei cu cel mai mic cost care formeaza cu precedentele muchii alese un arbore.Si acest algoritm se ócadreaza ón metoda Greedy.

<u>Algoritmul lui Prim ón pseudocod este:</u>

```
Program Prim
 Citeste n,m, E,C   {E- matricea muchiilor, C- vectorul costurilor}
 nar0               {contorizeaza nr APM obtinuti}
 for  i:= 1,n          {se genereaza arborele pornind de la nodul i}
      nod  r ‘  ;          {nod-vector ce memoreaza nodurile }
      cost r 0; j r 0   { se determina muchia de cost minim ce pleaca din i }
      repeat
            j r j+1
            i1 r E( j , 1 ); i2 r E{ j , 2 )
      until  ( i1 = i )  or ( i2 = i )  { i1, i2 sunt extremitatile muchiei }
      nod r nod 4 { i1 , i2 }
      cost r cost + C( j );  na r na +1;  arb ( na ) rj
      { alegerea celorlalte n-2 muchii}
      for  k=2,n-1
            j r 0
            repeat
                  j rj+1
                  i1 r E ( j, 1 ); i2 r E ( j, 2 )
            until ( i1 c nod ) xor ( i2 c nod )
            nod ρ nod 4 { i1, i2 }
            arb(na) r arb(na) 4 { j }
            { se testeaza daca s-a obtinut un arbore nou}
            j r0
            repeat
                  jrj+1
            until  arb(j) = arb(na)
            if  j < na  then   na r na-1
      stop .
```

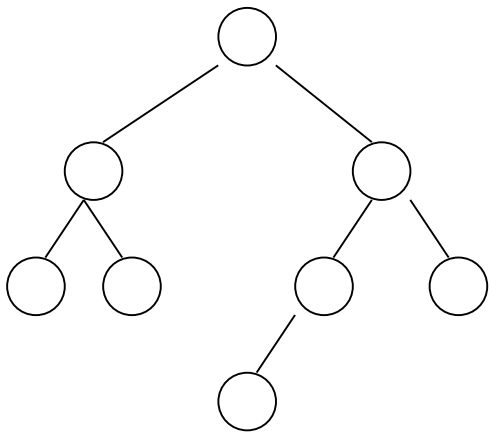
<u>3. ARBORI BINARI</u>

Definitie: Un **<u>arbore binar</u>** este un arbore în care fiecare nod are cel mult doi fii: fiul st, ng si fiul drept.

Daca se elimina radacina si legaturile ei se obtin doi arbori binari care se numesc subarborii st, ng si drept ai arborelui initial.

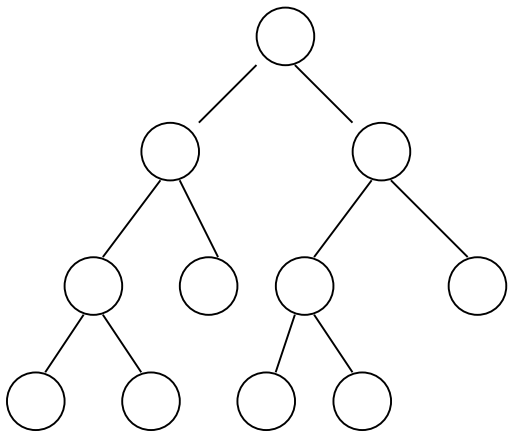
Deci arborele binar este o structura recursiva de date. Un arbore binar nevid fie se reduce la radacina , fie cuprinde radacina si cel mult doi subarbori binari.

Ex.:



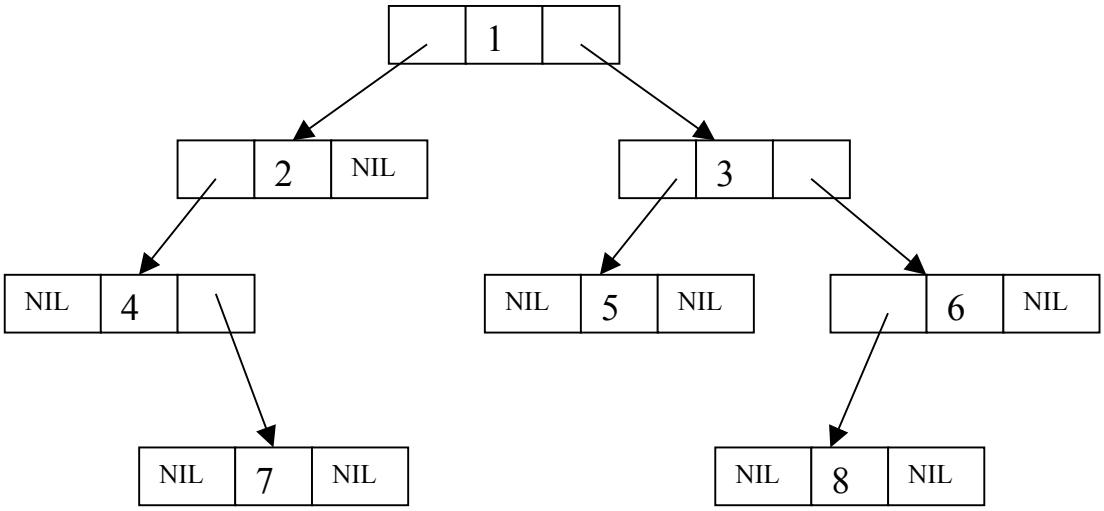
Se va face distinctie între fiul st, ng si fiul drept al arborelui

Definitie Un **<u>arbore binar complet</u>** este un arbore în care fiecare nod , care nu este terminal, are doi fii.



3.1. REPREZENTAREA ARBORILOR BINARI

- Un arbore binar se poate implementa foarte usor cu ajutorul adreselor de ólantuire, fiecare element cuprinz,nd , pe l,nga informatia propriu-zisa asociata nodului, adresa fiului st,ng si adresa fiului drept, acestea exprim,nd legaturile existente ótre arbori. Acest lucru se poate reprezenta grafic astfel:



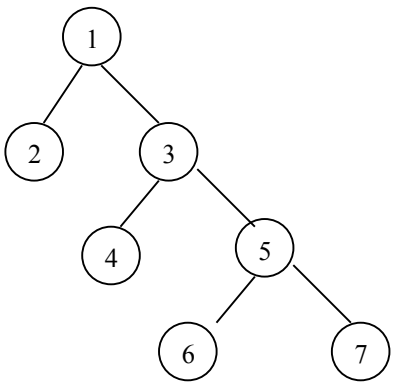
Declararea unei structuri de tip arbore se face astfel:

```
Type reper=^arbore;
        arbore= record
            inf : byte;
           st,dr: reper
        end;
var radacina : reper;
```

- O alta reprezentare se realizeaza *cu ajutorul vectorilor*.

Se specifica radacina si pentru fiecare v,rf se specifica descendentul st,ng si cel drept. Se folosesc doi vectori S si D, unde , pentru fiecare v,rf i, S[i] specifica descendentul st,ng, iar D[i] descendentul drept. INF [i] reprezinta informatia asociata nodului. S[i]=0 sau D[i]=0 semnifica lipsa unui descendent st,ng, respectiv a unui descendent drept.

Ex.: Fie arborele:



n = 7 RAD = 1

S=(2, 0 ,4 ,0 ,6 ,0 ,0)

D=(3, 0, 5, 0, 7, 0, 0)

- O alta reprezentare a arborilor binari se realizeaza d, ndu-se vectorii TATA care precizeaza, pentru fiecare v, rf i nodul TATA [i] si Desc care indica prin valoarea ñ1 sau 1 daca v, rful i este descendentul st, ng sau drept al parintelui sau TATA[i]. Pentru arborele de mai sus vom avea:

TATA = (0, 1, 1, 3, 3, 5, 5)
DESC = (0, -1, 1, -1, 1, -1, 1)

Elevii trebuie sa stie sa aleaga acel mod de reprezentare care ó avantajeaza ó rezolvarea fiecarei probleme ó parte.

Ei trebuie sa stie ca alocarea statica a arborilor este posibila atunci c, nd ó problema se cunoaste numarul maxim de noduri ale arborelui.

3.2. CREAREA SI TRAVERSAREA UNUI ARBORE BINAR

Crearea unui arbore binar se rezolva prin metoda *divide et impera* , descompun, nd problema ón trei subprobleme:

1. *crearea nodului radacina (dim=1);*
2. *crearea subarborelui st, ng;*
3. *crearea subarborelui drept.*

Descompunerea continua p, na se ajunge la problema crearii unui subarbore vid (dim=0) . Problema crearii unui subarbore vid se rezolva prin initializarea variabilei reper asociata radacinii arborelui cu constanta NIL . Rezolvarea unei probleme de dimensiune 1 (crearea unui nod radacina), presupune alocarea memoriei si initializarea zonei de informatii asociate nodului.

Daca cele trei probleme de dimensiuni mici sunt rezolvate, ón faza de combinare a solutiilor se leaga nodul radacina de cei doi subarbori. Œn acest fel s-a construit un subarbore mai mare care reprezinta solutia unei subprobleme din descompunere. Prin combinarea solutiilor se continua constructia arborelui.

Programul care creaza un arbore binar , prezentat ón continuare, utilizeaza o functie de creare numita *arbore* care returneaza adresa radacinii arborelui creat:

```
Program  creare_parcurgere_arbore_binar ;
type reper=^nod;
    nod=record
            inf:byte;
            st,dr:reper
        end;
var rad:reper;  {adresa radacinii arborelui}

function arbore:reper;  {returneaza adresa radacinii}
var  p:reper; val:byte;
begin
    read(val);
    if val>0 then    {subarborele este nevid}
      begin
        new(p);p^.inf:=val;{generarea radacinii p^ a subarborelui}
        write('nodul din stg. nodului ',val,':');
        p^.st:=arbore;{gen. subarb. st. si-l leaga de rad}
        write('nodul din dr. nodului ',val,':');
        p^.dr:=arbore;{gen. subarb. dr. si-l leaga de rad}
      end
         else p:=nil;
   arbore:=p;
end;
```

Parcurgerea arborilor binari consta în examinarea în mod sistematic a nodurilor sale a.Î fiecare nod sa fie atins o singura data. Traversarea arborelui presupune vizitarea fiecarui nod o singura data, operatie echivalenta cu o liniarizare a arborelui.

Exista trei modalitati importante de traversare a arborelui:

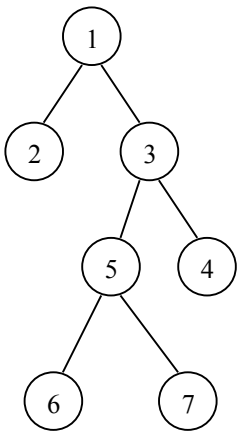
a) **Traversarea în preordine :** se viziteaza radacina, apoi , tot în preordine, se viziteaza nodurile subarborelui st, ng , apoi acelea ale subarborelui drept.
b) **Traversarea în inordine :** se viziteaza în inordine nodurile subarborelui st, ng, apoi radacina, si apoi , tot în inordine , nodurile subarborelui drept.
c) **Traversarea în postordine :** se viziteaza în postordine nodurile subarborelui st, ng, apoi , tot în postordine , nodurile subarborelui drept , si apoi radacina.

Parcurgerea arborilor este o problema care se poate descompune în trei subprobleme identice.Astfel, pentru parcurgerea în preordine , problema se descompune în urmatoarele subprobleme de dimensiuni mai mici:

a) *Parcurgerea radacinii* (un singur nod => dimensiunea=1)
b) *Parcurgerea subarborelui st, ng*;
c) *Parcurgerea subarborelui drept*.

Descompunerea subproblemelor b) si c) continua p, na c, nd se ajunge la subarbori vizi.

Fie arborele:



Parcurgerea acestuia furnizeaza v, rfurile în ordinea urmatoare:

preordine: 1, 2, 3, 5, 6, 7, 4
inordine: 2, 1, 6, 5, 7, 3, 4
postordine: 2, 6, 7, 5, 4, 3, 1

Consider, nd declaratiile arborelui creat mai sus , putem scrie urmatoarele proceduri:

```
procedure preordine(p:reper);
begin
   if p=nil then
     begin
        write(p^.inf:2);      {R}
        preordine(p^.st);     {S}
        preordine(p^.dr);     {D}
     end
end;
```

```
procedure inordine(p:reper);
begin
    if p=nil then
      begin
          inordine(p^.st);      {S}
          write(p^.inf:2);      {R}
          inordine(p^.dr);      {D}
      end
end;


 procedure postordine(p:reper);
begin
    if p=nil then
      begin

          postordine(p^.st);      {S}
          postordine(p^.dr);      {D}
          write(p^.inf:2);        {R}
      end
end;
```

Programul principal care apeleaza procedurile respective va fi:

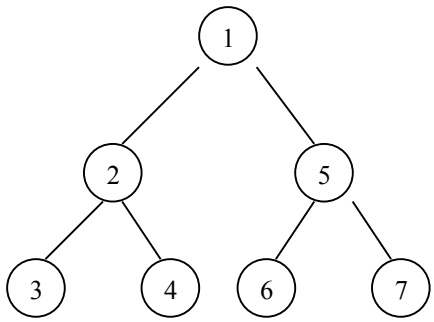
```
begin
    writeln('nodul radacina:');
    rad:=arbore;
    writeln;writeln('arborele parcurs in preordine: ');
    preordine(rad);
    writeln;writeln('arborele parcurs in inordine: ');
    inordine(rad);
    writeln;writeln('arborele parcurs in postordine: ');
    postordine(rad);
    writeln;readln
end.
```

3.3. APLICATII ALE ARBORILOR BINARI

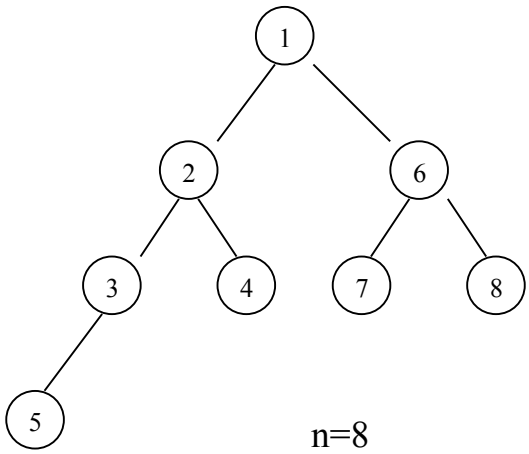
v *ARBORE PERFECT ECHILIBRAT*

Definitie *Se numaste* **arbore perfect echilibrat** *acel arbore care are diferenta dintre numarul v, rfurilor subarborilor oricarui v, rf din arbore egala cu cel mult 1.*

Ex



n=7



n=8

Conditia din definitia arborelui este respectata pentru un arbore cu n v, rfuri daca subarborele st, ng are *[n / 2]* v, rfuri , iar subarborele drept are *n ñ [n / 2] ñ 1* v, rfuri. Subprogramul de generare al unui astfel de arbore este urmatorul:

```
function arbore (n:byte) :reper ;{returneaza radacina subarborelui creat }
var p:reper;
   ns,nd:byte;
begin
    write('nodul: ');
    new(p);read(p^.inf);          {gen.rad. subarb.}
    ns:=n div 2;                  {gen subarb st cu ns noduri}
     if ns>0 then p^.st:=arbore(ns)
              else p^.st:=nil;
    nd:=n-ns-1;                   {gen subarb dr cu nd noduri}
    if nd>0 then p^.dr:=arbore(nd)
             else p^.dr:=nil;
    arbore:=p
 end;
```

v *ARBORI DE SORTARE*

Definitie *Se numeste* **arbore de sortare** *un arbore binar complet care are toate v, rfurile terminale pe un singur nivel sau pe doua nivele consecutive. V, rfurile terminale sunt asociate celor n elemente ale unui sir care se doreste a fi ordonat crescator.*

Arborii de sortare sunt folositi Ó sortarea elementelor unui sir printr-o metoda numita selectie arborescenta

Daca n este o putere a lui 2 , atunci arborele de sortare corespunzator are toate v, rfurile terminale pe acelasi nivel, altfel exista doua puteri consecutive ale lui 2, fie ele 2^r si 2^{r+1} a.Ó sa avem:

$$2^r < n < 2^{r+1}$$

Œ acest caz arborele de sortare corespunzator are v, rfurile pe nivelele consecutive *r* si *r+1*. Not, nd cu x numarul de v, rfuri terminale de pe nivelul r+1 si cu y numarul de v, rfuri terminale de pe nivelul r, avem urmatoarea relatie:

$$x + y = n$$

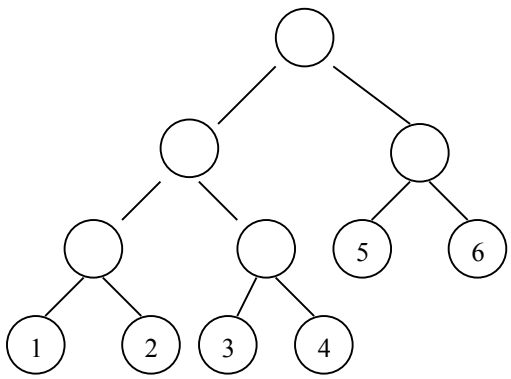
Pe de alta parte, daca , pornind de la arborele de sortare, fiecarui v, rf de pe nivelul r Ó asociem doi descendenti (pe nivelul r+1) am obtine un arbore binar complet cu toate v, rfurile terminale pe acelasi nivel (r+1), si, evident, am avea urmatoarea relatie:

$$x + 2 * y = 2^{r+1}$$

Grup, nd relatiile de mai sus obtinem un sistem de doua ecuatii cu doua necunoscute, dupa a carui rezolvare obtinem:

$$x = 2 * n - 2^{r+1}$$
$$y = n \tilde{n} x$$

De exemplu , pentru n=6 putem sa construim urmatorul arbore de sortare:



Elementele ce se vor sorta sunt plasate pe varfurile terminale de la st, nga la dreapta. Metoda de selectie arborescenta este inspirata din organizarea unui turneu prin eliminare .Completam de jos în sus v, rfurile interioare ale arborelui cu minimul valorilor asociate descendentilor. În v, rful radacina al arborelui se va gasi cel mai mic element din sir, el determin, ndu-se în urma a n-1 comparatii între elementele sirului. Extragem din radacina valoarea minima, pun, nd-o pe locul sau definitiv, adica în a[1].

Pentru elementele ramase vom executa urmatorul pas:

Fie lantul care uneste radacina cu v, rful terminal din care a provenit valoarea minima. În nodul parinte asociat lui z urca valoarea asociata v, rfului *frate* al lui. Se parcurge apoi lantul spre radacina si se corecteaza valoarea asociata fiecarui v, rf, calcul, ndu-se valoarea minima dintre valorile asociate descendentilor sai. În radacina va ajunge astfel elementul minim din elementele ramase în sir. Se memoreaza aceasta valoare în a[2] si se continua astfel p, na la extragerea din arbore a tuturor valorilor sale provenite din sirul initial.

Pentru a implementa în Pascal acest algoritm va trebui sa construim întâi arborele de sortare. Acest lucru îl realizam folosind o procedura recursiva care va construi un nod interior sau un nod terminal, în functie de nivelul pe care ne situam, iar daca suntem pe nivelul r, trebuie sa facem distinctie între cazul în care construim un nod interior, parinte al unuia dintre cele x noduri terminale de pe nivelul r+1 sau un nod terminal.

Va trebui sa calculam întâi valorile lui r , x si y . Evidenta construirii nodurilor terminale o tinem cu variabila z, iar a nodurilor neterminale (în numar de x/2) de pe nivelul r cu variabila u.

Scoaterea valorilor din arborele de sortare se face cu procedura recursiva *cauta,* care merge în ad, ncime p, na c, nd se identifica v, rful în care urca valoarea asociata descendentului pentru care valoarea fratelui sau este eliminata din arbore în etapa curenta.

Este important sa stim de unde vine valoarea în radacina, pentru a cunoaste locul în care se va introduce valoarea maxint si de unde va pleca în sus pentru completarea corecta a valorilor nodurilor interioare cu minimul valorilor celor doi descendenti.

Programul Pascal de sortare folosind selectia arborescenta va fi prezentat în capitolul de aplicatii.

v *ARBORELE BINAR DE CAUTARE*

Definitie *Se numeste* **arbore binar de cautare** *un arbore ale carui noduri cuprind o singura cheie de identificare, nodurile cu chei mai mici dec, t valoarea cheii asociate unui anumit nod se gasesc Ó subarborele st, ng al acestuia, iar nodurile ale caror chei au valori mai mari dec, t v se gasesc Ó subarborele sau drept.*

Cautarea unei informatii identificate printr-o valoare v a cheii Ó cepe de la radacina si se termina Ó cel mai rau caz la unul din nodurile terminale, cautarea presupun, nd testarea a cel mult at, tea noduri c, te niveluri are arborele binar de cautare. Dispunerea nodurilor arborelui pe niveluri face ca numarul testelor la cautare sa fie , Ó general, mai mic dec, t Ó cazul listelor ordonate.

In capitolul cu aplicatii se exemplifica utilizarea arborilor de cautare Ó rezolvarea unei probleme care se numeste ì *Construirea concordantelor* ì . Aceasta problema cere *sa se afiseze cuvintele distincte si frecventa lor de aparitie stiind ca se citesc mai multe cuvinte, c, te unul de pe o linie, p, na la CTRL / Z.*

Cuvintele citite vor fi puse Ó tr-un arbore binar de cautare astfel:

- Ó radacina se pune primul cuv, nt ;
- Ó st, nga ei se pune cuv, ntul urmator care-l precede alfabetic pe primul
- Ó dreapta se pune cuv, ntul urmator care-i urmeaza alfabetic s.a.m.d.

Fiecare nod din arbore cuprinde, Ó afara de cuv, ntul care reprezinta cheia de identificare a nodului si frecventa de aparitie si cele doua adrese de Ó lantuire.

Adaugarea unui cuv, nt Ó arbore presupune cautarea lui Ó structura arborescenta si legarea lui de nodul terminal corespunzator, daca cuv, ntul este diferit de cele existente deja.

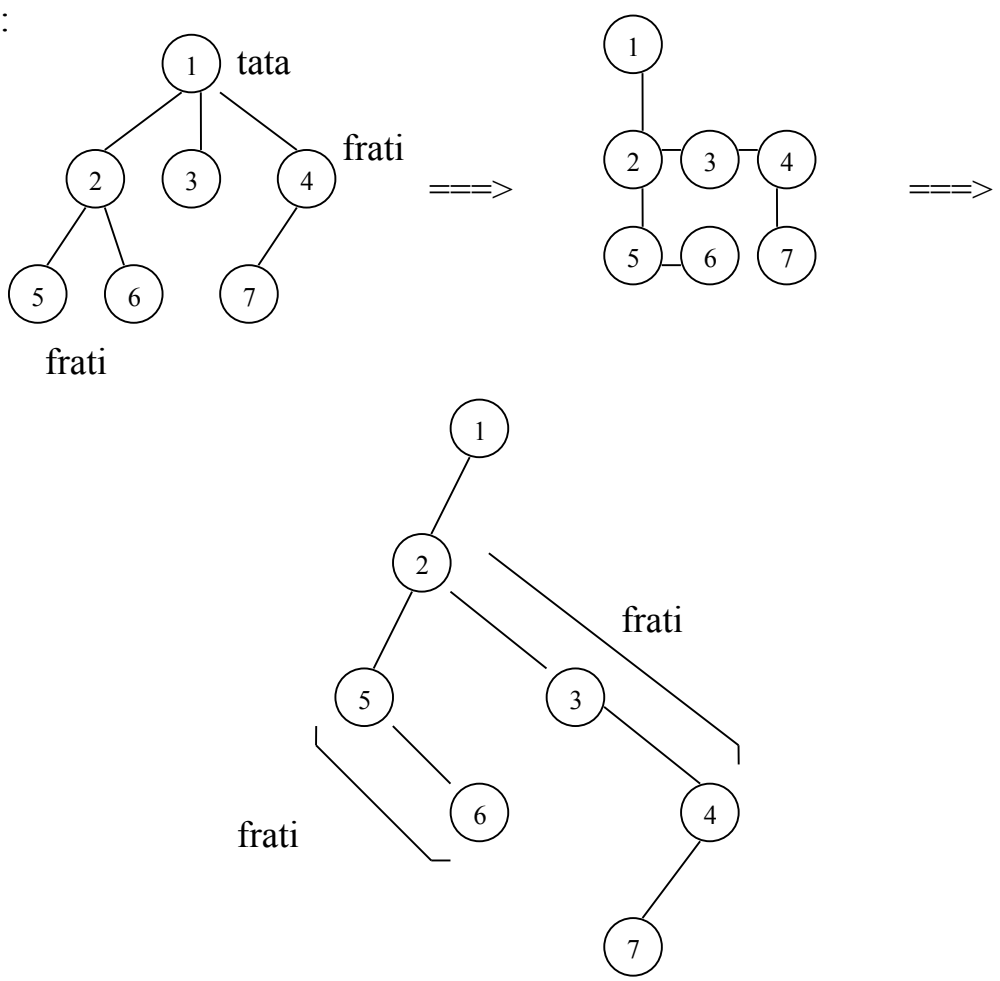
Afisarea cuvintelor se obtine Ó ordine alfabetica daca se traverseaza arborele de cautare Ó inordine. Daca cheile asociate unui arbore binar de cautare sunt numerice, traversarea acestuia Ó inordine parcurge nodurile Ó ordinea crescatoare a valorilor cheilor asociate.

∨ ***ARBORELE GENEALOGIC***

Arborele genealogic al unei persoane este un arbore oarecare , nodurile neterminale ale arborelui put, nd avea unul sau mai multi fii.

Un arbore oarecare poate fi reprezentat ca un arbore binar daca se pastreza legatura spre tata numai pentru primul fiu (legatura st, nga), iar ceilalti fii , fratii primului, se leaga ón tre ei (pe legatura dreapta).

Ex.:



Problema a carei rezolvare este prezentata ón capitolul cu aplicatii are urmatorul enunt:

Se considera un arbore genealogic care cuprinde descendentii unei persoane. Fiind date numele a doua persoane din familie , sa se afiseze cel mai apropiat stramos comun al celor doua persoane. Datele se considera corecte.

v *FORMA POLONEZA A EXPRESIILOR ARITMETICE*

Este una dintre cele mai importante aplicatii ale arborilor binari si a fost introdusa de matematicianul polonez J. Lukasiewicz.

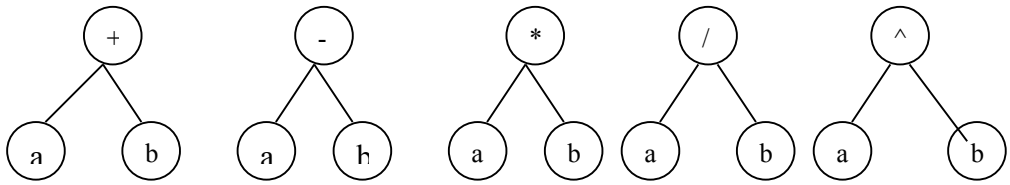
Se poate asocia unei expresii aritmetice E un arbore binar complet astfel:

- Unei expresii aritmetice formate dintr-un singur operand Ó asociem un arbore binar format doar din nodul radacina Ó în care punem operandul respectiv
- Daca expresia aritmetica E este de forma *E1 op E2 ,* unde op este unul dintre operatorii admisi , iar E1 si E2 sunt expresii aritmetice, arborele binar complet asociat are Ó în radacina operatorul op , ca subarbore st, ng arborele asociat expresiei E1, iar ca subarbore drept, arborele binar asociat expresiei E2.
- Daca E = (E1), unde E1 este o expresie aritmetica, atunci arborele binar asociat lui E coincide cu arborele binar asociat lui E1.

Ex.: Pentru expresiile aritmetice simple :

a+b , a-b , a*b , a/b , a^b

arborii binari asociati sunt:



Teorema *Parcurgerea Ó în preordine a unui arbore binar asociat unei expresii aritmetice E da forma prefixata a expresiei E.*

Definitie Fie o expresie aritmetica . Forma poloneza prefixata « a lui E se obtine astfel:

Ø , =a, pentru orice operand *a* care este o constanta sau o variabila

Ø $\overline{E1\ op\ E2}$ =op $\overline{E1\ E2}$, pentru orice expresii aritmetice E1 si E2

Ø $\overline{(E)}$= $\overline{E}$, pentru orice expresie aritmetica

Pentru expresia E= a*b+c/d-e, forma poloneza prefixata asociata este: -+*ab/cde
Œntr-adevar ,

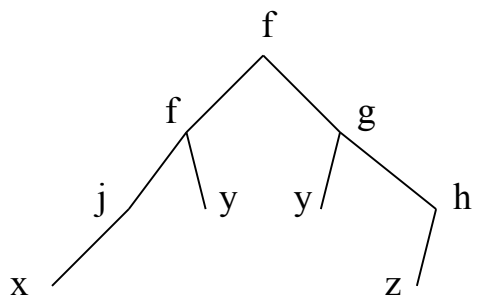
$$a*b+c/d-e=-a*b+c/de=-+a*bc/de=-+*ab/cde$$

*Obs.:*Forma poloneza asociata unei expresii aritmetice nu este unica.Programul care construieste un arbore binar asociat unei expresii si obt. formei prefixate va fi prezentat ón capitolul care contine aplicatii.

v ***REPREZENTAREA FUNCTIILOR COMPUSE***

Parcurgerea ón preordine a arborilor binari permite scrierea functiilor compuse fara paranteze. Pentru a ónelega acest lucru sa luam urmatorul exemplu:

Fie f si g doua functii de doua variabile si h si j doua functii de o variabila. Atunci functia compusa f(f(j(x),y),g(y,h(z))) se poate reprezenta prin urmatorul arbore binar:



Parcurgerea ón preordine a arborelui corespunde scrierii f f j x y g y h z care se realizeaza deci fara paranteze

CAP IV

GRAFURI ORIENTATE

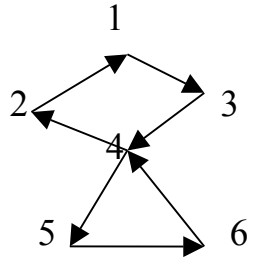
1. GENERALITATI:

Definitie Se numeste **graf orientat** o pereche ordonata de multimi (X, U), X fiind o multime finita si nevida de elemente numite noduri sau v, rfuri, iar U o multime de perechi ordonate de elemente distincte din X, numite arce.

Graful se noteaza cu G= (X , U)

Daca *x* si *y* sunt extremitatile arcului *u*, spunem ca *u* si *x* sunt *incidente*, la fel ca si *u* si *y* , iar v, rfurile *x* si *y* sunt *adiacente* Ó G. Daca exista arcul u=(x,y) , spunem ca x este extremitatea initiala, iar y este extremitatea finala a arcului.

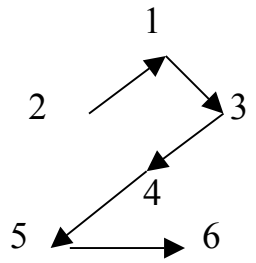
Ex:



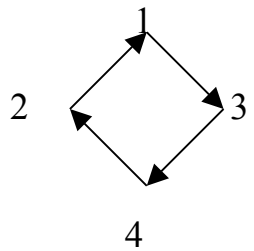
u1=(1,3), u2=(2,1), u3=(3,4), u4=(4,2)
u5=(4,5), u6=(6,4), u7=(5,6)

Definitie Un **graf partial** al grafului G= (X , U) este un graf G1= (X , U) a.Ó V U, adica G1 are aceeasi multime de v, rfuri ca G, iar multimea de arce V este , fie chiar U, fie o submultime a acesteia.

Un graf partial al unui graf dat se obtine pastr, nd aceleasi v, rfuri si elimin, nd o parte din arce. Fie graful de mai sus. Vom desena un graf partial al lui si un subgraf al lui:



graf partial



subgraf

Definitie Un **subgraf** al unui graf G= (X,U) este un graf H=(Y,V) a.Ó Y_X, iar V contine toate arcele din U care au ambele extremitati Ó Y.

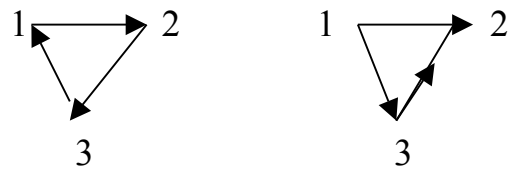
Vom spune ca subgraful H este indus sau generat de multimea de v, rfuri Y. Un subgraf se obtine elimin, nd o parte din v, rfuri si toate arcele incidente cu acestea.

Definitie Se numeste **graf complet** cu *n* v,rfuri un graf care are proprietatea ca orice doua v,rfuri distincte sunt adiacente.

Spre deosebire de grafurile neorientate la care exista un graf complet unic cu n v,rfuri , la grafurile orientate , pentru un n dat, putem construimai multe grafuri orientate av,nd n v,rfuri. Acest lucru decurge din faptul ca doua v,rfuri x, y sunt adiacente în mai multe situatii: exista arcul de la x la y,exista arcul de la y la x si exista ambele arce.

Exista , deci C^2_n posibilitati de a alege 2 noduri distincte, iar pentru acestea exista 3 situatii ca ele sa fie adiacente, rezulta ca , pentru un n dat sunt 3^x grafuri orientate complete cu n v,rfuri, unde x= C^2_n

Ex.: pentru n=3 avem 27 de grafuri orientate complete cu 3 v,rfuri



Pentru grafurile orientate sunt doua tipuri de arce incidente cu un v,rf: arce careî intraî si arce care ì iesî din acel v,rf. Se vor defini doua concepte diferite care sa corespunda numarului arcelor din fiecare din cele doua categorii:

- numim grad exterior al unui v,rf x , notat cu d^+ (x), numarul arcelor de forma (x,y)χU (numarul arcelor care ies din x).
- numim grad interior al unui v,rf x si-l notam cu d^- (x), numarul arcelor de forma (y,x)χU (numarul arcelor care intra în x).

Vom nota cu:

√$^+$ (x)={y χ X / (x,y)χ U} (multimea succesorilor lui x)

√$^-$ (x)={y χ X / (y,x)χ U} (multimea predecesorilor lui x)

Si cu

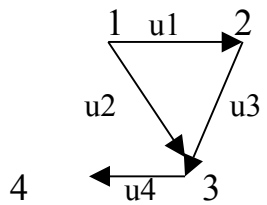
˘ $^+$(x)={u=(x,y) / u χ U } (multimea arcelor ce ies din x)

˘ $^-$(x)={u=(y,x) / u χ U } (multimea arcelor ce intra în x)

Definitie Se numeste **lant** într-un graf orientat un sir de arce cu proprietatea ca oricare doua arce vecine au o extremitate comuna.

Obs.:În definirea unui lant nu se tine cont de orientarea arcelor.

Ex.: Fie graful



L1=(u1, u3, u4)

L2=(u1, u2, u4)

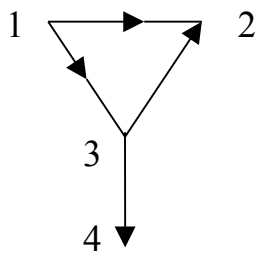
Definitie se numeste **drum** ßntr-un graf orientat G=(X, U),cu X= {x1, x2, Ö , xn} si se noteaza cu D, un lant ßn care toate arcele au aceeasi orientare .
Daca v, rfurile drumului sunt distincte , drumul se numeste **drum elementar**.

Definitie Se numeste **circuit** un drum care are x1=xn si arcele sunt distincte.Daca toate v, rfurile circuitului,cu exceptia primului si ultimului nod ,sunt distincte doua c, te doua, circuitul se numeste **circuit elementar**.

Definitie Un graf orientat se numeste **graf conex** daca pentru oricare doua v, rfuri distincte x,y χ X exista un lant de extremitati x,y

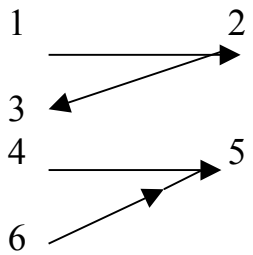
Definitie Se numeste **componenta conexa a unui graf orientat** ,G=(X,U), un subgraf conex al sau maximal ßn raport cu aceasta proprietate(oricare ar fi un nod din subfraf nu exista lant ßntre acel nod si v, rfurile care nu fac parte din subgraf*)*.
Ex.: Fie grafurile:

G=(X,U), unde X={1, 2, 3, 4} si U={ (1,2), (1,3), (3,2), (3,4)}



Graful G este conex
El are o singura componenta conexa ñ graful ßsusi

G1=(X1, U1) , unde X1={1, 2, 3, 4, 5, 6} si U1={(1, 2), (2, 3), (4, 5), (6, 5)}



Graful G1 nu este conex; el are doua componente conexe:
subgraful generat de multimea X1={1, 2, 3}
subgraful generat de multimea X2={4, 5, 6}

Definitie Un **graf** se numeste **tare conex** daca □x □=1 sau pentru oricare doua v, rfuri x, y χ X exista un drum de la x la y si un drum de la y la x.

Definitie **O componenta tare conexa** a unui graf G =(X,U) este un subgraf G1=(X1,Y1) al lui G care este tare conex si care este maximal ßn raport cu aceasta proprietate (oricare ar fi x χ X \ X1, subgraful lui G generat de X14 {x} nu mai este conex).

2. REPREZENTAREA GRAFURILOR ORIENTATE

Pentru reprezentarea unui graf orientat se folosesc:

1. Matrici asociate grafului:

1.1. Matricea de adiacenta este matricea cea mai utilizata în reprezentarea unui graf orientat. Aceasta este un tablou bidimensional cu n linii si n coloane av, nd elemente 0 sau 1 :

$$a_{ij} = \begin{cases} 1 & \text{daca } (x_i, x_j) \in U \\ 0 & \text{daca } (x_i, x_j) \notin U \end{cases}$$

Matricea de adiacenta nu mai este obligatoriu simetrica, ca în cazul grafurilor neorientate.

1.2. Matricea v, rfuri-arce- sau matricea de incidenta a grafului G - este o matrice $B=(b_{ij})_{nxm}$, unde:

$$b_{ij} = \begin{cases} 1 & \text{daca } x_i \text{ este extremitate initiala a arcului } u_j \\ -1 & \text{daca } x_i \text{ este extremitate finala a arcului } u_j \\ 0 & \text{daca } x_i \text{ nu este extremitate a arcului } u_j \end{cases}$$

1.3. Matricea drumurilor- se numeste matricea drumurilor asociata grafului G matricea $M= (m_{ij})_{nxm}$ data prin:

$$m_{ij}= \begin{cases} 1 \text{ , daca exista drum de la xi la xj în G} \\ 0 \text{ , în caz contrar} \end{cases}$$

Un algoritm simplu de determinare a matricii drumurilor unui graf este algoritmul Roy-Warshall. Acest algoritm construieste matricea drumurilor pornind de la matricea de adiacenta a grafului si consta în urmatoarele:

```
for k=1 to n
  for I=1 to n (i!k)
    for j=1 to n (j!k)
        if a[I,j]=0 then a[I,j]:=min(a[i,k], a[k,j])
        endif
    endfor
  endfor
endfor
```

Asadar un element al matricii a[i,j]=0 al matricii A devine 1 daca exista un nod k astfel ó c, t a[i,k]=1 si a[k,j]=1 deci c, nd exista drum de la xi la xk si drum de la xk la xj.

Programul Pascal de determinare a matricii drumurilor unui graf G=(X,U) cu X={1,..,n} este urmatorul:

```
Program rw;
Const nmax=100;
Var a:array[1..nmax,1..nmax] of 0..1;
        n,i,j,k: 1..nmax; m:word;

procedure init;  {initializeaza matricea de adiacenta}
var i,x,y:1..nmax;
        begin
                writeln(ëdati nr.de noduri:í); readln(n);
                writeln(ëdati nr. de arce:í);readln(m);
                for i:=1 to m do
                begin
                        write(ëarcul ë, i,í:í);
                        readln(x,y);
                        a[x,y]:=1;
                end;
        end;

begin
        init;
        for k:=1 to n do
          for i:=1 to n do
            for j:=1 to n do
                if a[i,j]=0 then a[i,j]:=a[i,k]*a[k,j];
        writeln(ëmatricea drumurilor este:í)
        for i:=1 to n do
           begin
             writeln;
             for j:=1 to n do write(a[i,j]:3);
          end
end.
```

2.Liste de adiacenta- pentru orice v, rf xχ X se construieste o lista L(x) a succesorilor sai. Listele de adiacenta pot fi reprezentate folosind tablouri sau structuri dinamice de date :

2.1. Reprezentarea listelor de adiacenta folosind tablouri:

Se considera un tablou T:array{1..2,1..n] of integer si un vector C:array[1..n] of integer cu urmatoarea semnificatie:

- C[i]- indica coloana din T din care Ó incepe lista vecinilor lui i;
- T[1,i] -reprezinta un v, rf;
- T[2,i]- reprezinta adresa (coloana pe care se afla) urmatorul element din lista Ó in care se afla T[1,i]; daca T[1,i] este ultimul element din lista, T[2,i]=0

2.2. Reprezentarea listelor folosind structuri dinamice de date:

Se considera un vector C cu n componente de tip referinta; C[i] pointeaza la Ó inceputul listei de adiacenta a lui i. Sunt deci necesare urmatoarele declaratii:

```
Type reper=^element;
        Element=record
                nod:1..n;
                next:reper
        end;
var C:array[1..n] of reper;
        p: reper;
```

Initial for i:=1 to n do C[i]:=nil;

Existenta unui arc (i,j) presupune actualizarea

```
new(p);
p^.nod:=j;
p^.next:=C[i];
C[i]:=p;
```

3. DRUMURI MINIME SI MAXIME ÎN GRAFURI ORIENTATE

Fie un graf orientat G=(X, U) si o functie l: U τR_+ , care asociaza fiecarui arc uχ U lungimea (costul sau ponderea) sa cu l(u).

Lungimea unui drum în acest graf este egala prin definitie cu suma lungimilor asociate arcelor sale. Pentru graful considerat se pot pune urmatoarele probleme:

- ***Determinarea drumurilor de lungime minima (maxima) între doua v, rfuri xi, xj;***
- ***Determinarea drumurilor de lungime minima (maxima) pornind dintr-un v, rf dat la toate celelalte noduri;***
- ***Determinarea drumurilor de lungime minima (maxima) între oricare doua v, rfuri.***

În continuare se vor prezenta algoritmi de obtinere a unor astfel de drumuri fara a determina toate drumurile dintre v, rfurile grafului.

Rezolvarea problemei determinarii unor drumuri de lungime minima într-un graf are multiple aplicatii practice. Daca vom considera drept noduri diferite puncte dintr-un oras, not, nd ponderea cu l(u) , unde u=(xi,xj) reprezinta durata de trecere de la xi la xj, problema revine la determinarea drumurilor de durata minima; daca l(u) reprezinta costul transportului de la xi la xj problema revine la determinarea drumurilor av, nd costul de transport minim, etc.

Pentru tratarea problemelor de minim vom asocia grafului G o matrice a costurilor $C=(c_{ij})_{nxn}$ definita astfel:

$$c_{ij} = \begin{cases} 0 & \text{daca } i=j \\ l(xi,xj), & \text{daca } (xi,xj)\chi U \\ +\square, & \text{daca } (xi,xj)\forall U \end{cases}$$

Semnificatia acestei alegeri este urmatoarea: drumul cel mai scurt de la xi la el însusi este de lungime 0, iar inexistenta arcului (xi, xj) este echivalenta cu existenta unui arc de lungime infinita care nu va interveni niciodata într-un eventual drum minim de la xi la xj.

Se vor prezenta în continuare urmatorii algoritmi:

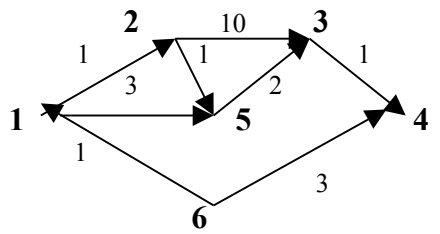
- ***Algoritmul lui Dijkstra***
- ***Algoritmul lui Roy-Floyd***
- ***Metoda drumului critic***

În capitolul cu aplicatii vor fi prezentati algoritmi în limbajul Pascal care ilustreaza utilizarea acestor algoritmi. Parcurgerea grafurilor orientate se realizeaza prin aceleasi metode care au fost prezentate la grafurile neorientate : DF si BF.

4.1. ALGORITMUL LUI DIJKSTRA

Problema: *Fiind dat un graf orientat G=(X,U), o functie l: Ut R_+ si un nod x_0 , sa se determine toate v, rfurile x_i pentru care exista drum de la x_0 la x_i , lungimea celui mai scurt drum ,precum si unul dintre drumurile minime de la x_0 la x_i.*

Algoritmul utilizeaza metoda Greedy gener,nd drumurile minime Ó ordinea crescatoare a lungimilor.
Ex.:



Pentru graful considerat , plec,nd din nodul 1, vom obtine Ó ordine:

D1=(1, 2) de lungime 1; (drumul de la 1 la 2 are lungimea 1);
D2=(1, 2, 5) de lungime 2; (drumul de la 1 la 5 are lungimea 2);
D3=(1, 2, 5, 3) de lungime 4; (drumul de la 1 la 3 are lungimea 4);
D4=(1, 2, 5, 3, 4) de lungime 5 (drumul de la 1 la 4 are lungimea 5);
De la 1 la 6 nu exista drum.

Se considera S multimea v,rfurilor xiχ X pentru care am gasit drum minim de la x_0 la x_j . Initial S={ x_0 }. La fiecare pas adaugam Ó S acel nod x_k χ X \ S cu proprietatea ca drumul minim de la x_0 la x_k are cel mai mic cost dintre drumurile de la x_0 la x_p , cu x_pχ X \ S. Pentru a alege nodul x_k ce urmeaza a fi adaugat Ó S vom folosi un vector d=(d1,d2,Ö ,dn) a.Ó

$$di = \begin{cases} \text{lungimea drumului minim de la } x_0 \text{ la } x_i, \text{ daca } x_j \chi S \\ \text{lungimea drumului minim de la } x_0 \text{ la } x_i, \text{ ce foloseste numai v,rfuri din S,} \\ \qquad \text{daca } x_i \forall S \end{cases}$$

Initial di = C(i_0, i) … i= 1, n unde C este matricea costurilor. Dupa adaugarea lui x_k Ó S trebuie actualizate valorile lui d. Œ final vectorul *d* va contine costurile (lungimile) drumurilor minime de la x_0 la celelalte noduri; daca pentru un nod x_j nu exista un astfel de drum , dj=□. Pentru a retine si drumurile minime (nu numai lungimile lor) vom considera un vector PREC care retine indicele precedentului fiecarui nod Ó drumul minim de la x0 la acel nod si care se actualizeaza la fiecare modificare dj:=dk+C(k,j) PREC:=k. Algoritmul se Ó cheie c,nd S contine toate nodurile pentru care exista drum de la nodul de plecare . Algoritmul Ó Pascal este prezentat Ó capitolul cu aplicatii.

4.2. ALGORITMUL LUI ROY-FLOYD

Problema: *Fiind dat un graf G=(X,U) cu X={x_1, x_2 ,Ö , x_n } si o functie l: UtR, sa se determine pentru fiecare pereche de noduri x_i , x_j (i!j) lungimea minima a drumurilor de la x_i la x_j precum si aceste drumuri (ó caz ca exista drum de la x_i la x_j).*

Algoritmul Roy-Floyd determina lungimile minime ale drumurilor ótre oricare doua noduri ale grafului ótr-o matrice C= (c_{ij})$_{nxn}$ unde

$$c_{ij} = \begin{cases} 0, \text{ daca } i=j \\ \text{lungimea drumului minim de la } x_i \text{ la } x_j \text{ , daca exista drum de la } x_i \text{ la } x_j \\ \square \text{ daca nu exista drum de la } x_i \text{ la } x_j \end{cases}$$

Determinarea matricii C este asemanatoare algoritmului Roy-Warshall pentru obtinerea matricii drumurilor. Se porneste de la matricea costurilor C :

```
for k=1 to n
  for i=1 to n   (i!k)
    for j=1 to n    (j!k)
        cij :=min ( cjj ,  cik + ckj )
    endfor
  endfor
endfor
```

Acest algoritm poate fi privit ca o succesiune de n transformari aplicate matricii C. Simultan cu determinarea lungimilor minime ale drumurilor pot fi retinute si acestea. Se va folosi o matrice D (patratica , av,nd nxn elemente) ale carei elemente d_{ij} sunt multimi de noduri (d_{ij} va reprezenta ó final multimea nodurilor ce pot precede pe xj ó drumul minim de la xi la xj).

Acest algoritm de determinare a tuturor drumurilor minime ótre oricare doua v,rfuri ale unui graf G va fi prezentat ó capitolul cu aplicatii.

Pentru determinarea drumurilor de lungime maxima se ataseaza grafului o matrice M=(m_{ij})$_{nxn}$ definita astfel:

$$m_{ij} = \begin{cases} l(xi, xj) \text{ , daca } (xi,xj) \chi U \\ - \square \text{ , daca } (xi,xj) \forall U \text{ si } (i!j) \\ 0 \text{ , daca } i=j \end{cases}$$

Aceasta matrice este asemanatoare matricii costurilor atasata grafului pentru determinarea drumurilor minime, cu diferenta ca , pentru perechi de noduri xi, xj (i!j) pentru care nu exista arcul (xi,xj) , marcam în matrice - □.

Algoritmii de determinare a drumurilor minime pot fi adaptati cu mici modificari pentru determinarea drumurilor maxime.
Fie matricea M asociata grafului , iar $D=(d_{ij})_{nxn}$ cu

$$d_{ij} = \begin{cases} \{xi\} & \text{daca } m_{ij} > - \square \text{ si } (i!j) \\ ‘ & \text{daca } m_{ij} = - \square \text{ sau } i=j \end{cases}$$

```
for  k=1 to n
  for i=1 to n (k!I)
    for j=1 to n (k!j)
        if m ij < m ik + m kj then  m ij := m ik + m kj
                                    d ij := d kj
                              else
                                    if   m ij = m ik + m kj
                                    then   d ij := d ij 4 d kj
                                    endif
        endif
      endfor
    endfor
  endfor
```

În matricea M vom avea în final lungimile drumurilor maxime între oricare doua noduri, iar în d_{ij} … I,j χ {1, Ö ,n} vom avea multimea nodurilor ce pot precede pe xj într-un drum maxim de la xi la xj.

În capitolul cu aplicatii va fi prezentat algoritmul de determinare a tuturor drumurilor maxime între oricare doua noduri folosind acest algoritm

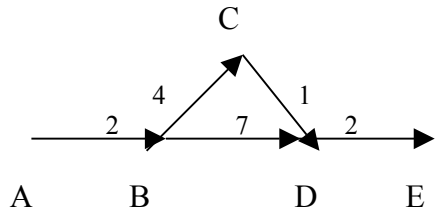
4. DRUM CRITIC ÎNTR-UN GRAF DE ACTIVITATI

Definitie : *Un* **graf de activitati** *este un graf asociat unei lucrari complexe a carei realizare presupune desfasurarea mai multor actiuni(procese, activitati).*

Pentru un astfel de graf nodurile reprezinta *evenimente* care pot fi interpretate ca indicând realizarea unor obiective partiale ale lucrarii ; arcele reprezinta *etapele* sau *activitatile elementare* ale lucrarii, iar lungimea asociata unui arc ñtimpul de desfasurare *a activitatii* asociate acelui arc.

Pentru o activitate, momentul începerii ei este reprezentat de extremitatea initiala, iar momentul terminarii ei de extremitatea finala a arcului respectiv.

Atunci când construim un astfel de graf trebuie sa respectam urmatoarea regula:
toate arcele care pleaca dintr-un vârf X reprezinta activitati ce nu pot începe decât dupa terminarea tuturor activitatilor reprezentate de arcele care sosesc în vârful X.
Ex.:



Pentru graful de mai sus începerea lucrarii este reprezentata prin vârful A, iar terminarea ei prin vârful E. Nodul B reprezinta evenimentul ce marcheaza terminarea activitatii reprezentate de arcul (A,B) si începerea activitatii reprezentate de arcele (B,C) si (B,D); nodul C este evenimentul ce marcheaza terminarea activitatii reprezentate de arcul (B,C) si începerea activitatii reprezentate de arcul (C,D) etc. Activitatile reprezentate de arcele (B,C) si (C,D) se desfasoara în acelasi timp cu activitatea reprezentata de arcul (B,D).

Pentru a nu avea mai multe arce paralele si de acelasi sens între doua vârfuri ale grafului se introduc evenimente si activitati fictive.

Definitie : *se numeste* **drum critic într-un graf de activitati** *un drum de lungime maxima care leaga nodul initial de nodul final.*

Pentru un graf dat drumul critic nu este unic Drumul critic reuneste activitati a caror întârziere duce la întârzierea întregii lucrari, activitati ce se numesc *activitati critice.* Calculul timpului de realizare a evenimentului final revine la cautarea în graf a drumului cel mai lung sau a drumului critic.

Metoda este cunoscuta sub denumirile: PERT- Programme Evaluation and Research Task si CPM- Critical Path Method.

Pentru fiecare eveniment putem vorbi de o data asteptata si de o data limita de realizare a sa.

CAP V

APLICATII

LISTE:

1. Teancuri de carti (STIVA):

Biblioteca unui liceu primeste un numar de carti si vrea sa-i separe pe autori. Scrieti un program care sa simuleze acesta activitate stiind ca autorii si titlurile cartilor se introduc de la tastatura p, na la CTRL\Z. Pentru fiecare autor se vor afisa titlurile cartilor si numarul de exemplare primit.

```
program carti;
type reper=^carte;
     carte=record
           titlu:string;
           n_ex:byte;
           next:reper
           end;
var i,n:byte; top:array[1..100] of reper;
    autor:array[1..100] of string; autorul,cartea:string;
function indice(autorul:string):byte;
var i:byte;
begin
  for i:=1 to n do if autor[i]=autorul then
              begin indice:=i;exit end;
  inc(n);autor[n]:=autorul;indice:=n
end;
procedure adauga(i:byte;cartea:string);
var p:reper;
begin
     p:=top[i];
     while p<>nil do
           if p^.titlu=cartea then begin
                              inc(p^.n_ex);exit
                              end
                              else p:=p^.next;
     new(p);
     p^.titlu:=cartea; p^.n_ex:=1;
     p^.next:=top[i]; top[i]:=p
 end;
 procedure listare(top:reper;autor:string);
 var p:reper;
 begin
      writeln('cartile ',autor);
      p:=top;
      while p<>nil do
            begin writeln(p^.titlu,' ',p^.n_ex);
            p:=p^.next  end
 end;
 begin
      for i:=1 to 100 do top[i]:=nil;
      n:=0;write('autorul: ');
      while not eof do
        begin
            readln(autorul);i:=indice(autorul);
           write('cartea:');readln(cartea);
           adauga(i,cartea);write('autorul:')
```

```
        end;
           for i:=1 to n do listare(top[i],autor[i])
    end.
```

2. Dispecerizare locomotive (COADA):

Se considera un depou de locomotive cu o singura linie de cale ferata cu o intrare si o iesire.Sa se scrie programul care realizeaza dispecerizarea locomotivelor prelucr, nd comenzi de forma: I-intrarea unei locomotive, E-iesirea unei loc, L-listarea si S- oprirea programului , afis, nd locomotivele existente.

```
program depozit;
  uses crt;
  type  reper =^element;
       element=record
         cod:word;
         next:reper
       end;
  var  prim, ultim, p:reper;
       comanda:char;
  procedure intrare;      {adaugarea unui element}
  var codul:word;
  begin
       write('codul locomotivei: ');readln(codul);
       if prim=nil
          then                {coada este vida}
              begin
               new(prim);
               prim^.cod:=codul;
               prim^.next:=nil;
               ultim:=prim;
               end
          else           {se adauga in coada un nou element}
             begin
               new(p);
               p^.cod:=codul;
               p^.next:=nil;
               ultim^.next:=p;
               ultim:=p
             end
  end;
  procedure iesire;  {stergerea primului element}
  begin
  if prim=nil then writeln('depoul este gol')
              else begin
                   writeln('iese locomotiva cu codul ',prim^.cod);
                   p:=prim;
                   prim:=prim^.next;
                   dispose(p)
                   end
  end;
  procedure listare;   {traversarea cozii}
  begin
       if prim=nil then writeln('depoul este gol')
                   else begin
                        writeln('locomotivele din depou sunt: ');
                        p:=prim;
                        repeat
                              writeln(p^.cod);
                              p:=p^.next
                        until p=nil
                        end
  end;
```

```
Begin
     clrscr;
     prim:=nil;
     repeat
           write('comanda: '); readln(comanda);
           comanda:=UpCase(comanda);
           case comanda of
             'I':intrare;
             'E':iesire;
             'S','L':listare;
             end
     until comanda='S' ;
     readln
end.
```

3 Jocul lui Joseph (LISTA CIRCULARA):

Un numar dat n de copii stau ó cerc si numara , ócep, nd cu primul copil asezat ó cerc si continu, nd ó ordinea initiala ó care au fost asezati .Fiecare al m-lea copil iese din cerc. C, stiga jocul ultimul copil ramas ó cerc.Sa se simuleze jocul stiind n,m s numele copiilor.

```
program Joseph;
type adresa=^copil;
     copil=record
        nume:string;
        next:adresa
     end;
var sant, ultim, p:adresa;
    n,m,i:byte;

Begin
     new(sant);ultim:=sant;    {lista vida}
     write('n=');readln(n);
     write('m=');readln(m);
     writeln('Copiii:');
     for i:=1 to n do
         begin
              new(p);
              readln(p^.nume);
              ultim^.next:=p;
              ultim:=p
         end;
     ultim^.next:=sant^.next;  {coada se transforma in lista liniara}
     {simulam jocul}
     p:=sant;
     repeat
           for i:=1 to m-1 do p:=p^.next;  {numaram m copiii}
           writeln('Iese copilul ',p^.next^.nume);
           p^.next:=p^.next^.next   {stergem logic elementul p^.next^}
     until p=p^.next;
     writeln(' Castiga ',p^.nume);
     readln
     end.
```

4. (LISTA DUBLU ŒLANTUITA ñ cu santinele):

Se citesc de la tastatura mai multe nr naturale nenule <=35000, mai putin ultimul element care este 0. Folosind o lista alocata dinamic :

a) sa se afiseze numerele ó ordinea citirii;

b) *sa se afiseze nr ó ordinea inversa citirii;*

c) *sa se afiseze nr ó ordine crescatoare.*

```
program numere;
type reper=^element;
     element=record
        info:word;
        pred, next:reper
     end;
var sant1,sant2,p,q:reper;
    n,sw:word;

 begin
      new(sant1);new(sant2);
      sant1^.next:=sant2;
      sant2^.pred:=sant1;
      write('numar: ');readln(n);
      while n<>0 do
            begin
                  p:=sant2;new(sant2);
                  p^.info:=n;
                  p^.next:=sant2;
                  sant2^.pred:=p;
                  write('numar: ');readln(n);
            end;
      writeln('a)');
      p:=sant1^.next;
      if p=sant2 then writeln('lista vida')
                 else begin
                      repeat
                            write(p^.info,' ');
                            p:=p^.next;
                      until p=sant2;
                      writeln
                      end;

      writeln('b)');
      p:=sant2^.pred;
      if p=sant1 then writeln ('lista vida')
                 else begin
                      repeat
                            write(p^.info,' ');
                            p:=p^.pred;
                      until p=sant1;
                      writeln
                      end;
      sw:=1;
      while sw=1 do
            begin
                  sw:=0;
                  p:=sant1^.next;
```

```
            if p<>sant2 then


                begin
                    repeat
                     if p^.info>p^.next^.info then
                         begin
                              sw:=1;n:=p^.info;
                              p^.info:=p^.next^.info;
                              p^.next^.info:=n
                         end;
                         p:=p^.next
                     until p=sant2^.pred;
                   end
      end;

write('c)');
p:=sant1^.next;
if p=sant2 then writeln('lista vida')
           else begin
                repeat
                      write(p^.info,' ');
                      p:=p^.next;
                until p=sant2;
                writeln
                end;
readln
end.
```

5. Stergerea elementului minim dintr-o lista (LISTA SIMPLU ÓNLANTUITA):

Sa se creeze o lista de numere óntregi distincte, citite de la tastatura(p, na la CTRL\Z). Sa se determine valoarea minima din lista si sa se stearga elementul corespunzator. Lista se va afisa óinainte si dupa stergere.Numerele se pun óin lista óin ordinea citirii.

```
program min_lista;
type reper=^element;
     element=record
        n:integer;
        next:reper
     end;
var prim,ultim,p:reper;
    nr:integer;

procedure adauga(nr:integer);
var q:reper;
begin
     p:=prim;
     while p^.n<>nr do p:=p^.next;     {cautam nr in lista}
     if p=nil then begin new(q);
                         q^.n:=nr;
                         ultim^.next:=q;
                         ultim:=q
end;
```

```
function adr_ant_min:reper;
var p,pmin:reper;
    min:integer;
begin
     min:=Maxint;
     p:=prim;
     while p<>nil do

             begin
                if p^.next^.n<min then
                   begin
                        min:=p^.next^.n;
                        pmin:=p
                   end;
                p:=p^.next
           end;
     writeln('minimul este ',pmin^.next^.n);
     adr_ant_min:=pmin
end;

procedure stergere(p:reper);
var q:reper;
begin
     q:=p^.next;        {salvam adresa elementului care se sterge}
     p^:=q^;
     Dispose (q)         {elibereaza memoria}
end;

procedure listare;
begin
     writeln('lista:');
     p:=prim;
     while p<>nil do
           begin
                write(p^.n,' ');
                p:=p^.next
           end;
     writeln
end;

Begin
     new(prim);
     write('primul nr este:');readln(nr);
     prim^.n:=nr;
     prim^.next:=nil;
     ultim:=prim;
     writeln('adauga numerele:');
     while not eof do
           begin
                read(nr);
                adauga(nr)
           end;
     listare;
     p:=adr_ant_min;{adresa elem anterior celui minim}
     stergere(p);
     listare;
     readln
end.
```

5. ***LISTA SIMPLU CHEALANTUITA*** cu santinela:

Se citeste un numar natural ce contine maxim 200 cifre. Sa se alcatuiasca un program care creeaza o lista simplu ólantuita ce contine ca elemente cifrele pare ale numarului dat. Programul va afisa elementele listei si suma elementelor sale.

```
program lista_sant;
uses crt;
type lista=^element;
     element=record
        info:integer;
        next:lista
        end;
var nr:string;
  suma,cifra,i,er:integer;
  sant,ultim,p:lista;

begin
     clrscr;
     write('numarul este:');readln(nr);
     new(sant);
     ultim:=sant;
     ultim^.next:=nil;
     suma:=0;
     for i:=1 to length(nr) do
         begin
           val(nr[i],cifra,er);
           if cifra mod 2=0 then
              begin
                suma:=suma+cifra;
                new(p);
                p^.info:=cifra;
                ultim^.next:=p;
                p^.next:=nil;
               end
         end;
     p:=sant;
     while p<>nil do
          begin
               write(p^.info,' ');
               p:=p^.next
            end;
        writeln('suma este: ',suma);
        readln
end.
```

ARBORI:

1. Crearea si traversarea unui arbore binar:

```
program parcurg_arbore;
type reper=^nod;
     nod=record
               inf:byte;
               st,dr:reper
         end;
var rad:reper;  {adresa radacinii arborelui}
function arbore:reper;  {returneaza adresa radacinii}
var p:reper;   val:byte;
begin
     read(val);
     if val>0 then    {subarborele este nevid}
      begin
           new(p);p^.inf:=val;{generarea radacinii p^ a subarborelui}
```

```
            write('nodul din stg. nodului ',val,':');
            p^.st:=arbore;{gen. subarb. st. si-l leaga de rad}
            write('nodul din dr. nodului ',val,':');
            p^.dr:=arbore;{gen. subarb. dr. si-l leaga de rad}
       end
          else p:=nil;
       arbore:=p;
end;
procedure preordine(p:reper);
begin if p=nil then
        begin
             write(p^.inf:2);      {R}
             preordine(p^.st);     {S}
             preordine(p^.dr);     {D}
        end
end;
procedure inordine(p:reper);
begin if p=nil then
        begin
             inordine(p^.st);      {S}
             write(p^.inf:2);      {R}
             inordine(p^.dr);      {D}
        end
end;
procedure postordine(p:reper);
begin  if p=nil then
        begin

             postordine(p^.st);     {S}
             postordine(p^.dr);     {D}
             write(p^.inf:2);       {R}
        end
end;
begin
     writeln('nodul radacina:');
     rad:=arbore;
     writeln;writeln('arborele parcurs in preordine: ');
     preordine(rad);
     writeln;writeln('arborele parcurs in inordine: ');
     inordine(rad);
     writeln;writeln('arborele parcurs in postordine: ');
     postordine(rad);
     writeln;readln end.
```

2. Arbore binar de cautare:

```
program concordante;
type reper=^nod;
     nod=record
       cuvant:string[20];
       frecv:byte;
       st,dr:reper
     end;
var rad:reper;
    cuv:string[20];

procedure inordine(p:reper);
begin
     if p<>nil then
        begin
             inordine(p^.st);
```

```
            writeln(p^.cuvant,' ',p^.frecv);
            inordine(p^.dr)
        end
end;
procedure caut_adaug(var p:reper);
begin
     if p=nil then
        begin
          new(p);
          with p^ do
             begin
                  cuvant:=cuv;
                  frecv:=1;
                  st:=nil;
                  dr:= nil
             end
        end
        else if cuv=p^.cuvant
             then inc(p^.frecv)
             else if cuv<p^.cuvant then caut_adaug(p^.st)
                                else caut_adaug(p^.dr)
end;

begin
     rad:=nil;
     writeln('Cuvintele :');
     while not eof do
           begin
                readln(cuv);
                caut_adaug(rad);
           end;
     if rad =nil then writeln('Arbore vid')
                 else begin
                      writeln('Cuvintele cu frecventele lor: ');
                      inordine(rad)
                      end;
     readln
end.
```

3. Crearea unui arbore perfect echilibrat:

```
program creare;
type reper=^nod;
     nod=record
       inf:byte;
       st,dr:reper
     end;
var rad:reper;
    n:byte;

function arbore(n:byte):reper;
var p:reper;
    ns,nd:byte;
begin
     write('nodul: ');
```

```
     new(p);read(p^.inf);{gen.rad. subarb.}
     ns:=n div 2;  {gen subarb st cu ns noduri}
     if ns>0 then p^.st:=arbore(ns)
             else p^.st:=nil;
     nd:=n-ns-1;   {gen subarb dr cu nd noduri}
     if nd>0 then p^.dr:=arbore(nd)
             else p^.dr:=nil;
     arbore:=p
end;

procedure preordine(p:reper);
begin
     if p<>nil then
        begin
             write(p^.inf:3);
             preordine(p^.st);
             preordine(p^.dr);
        end
                else write('.':2);
end;

begin
     write('n= ');readln(n);
     if n>0 then
        begin
             rad:=arbore(n);
             writeln('Arborele perfect echilibrat: ');
             preordine(rad)
        end
           else writeln('Arbore vid');
end.
```

4. Arbore genealogic

Se considera un arbore genealogic care cuprinde descendentii unei persoane. Fiind date numele a doua persoane din familie , sa se afiseze cel mai apropiat stramos comun al celor doua persoane.

```
Program arbore_genealogic;
uses crt;
type reper=^nod;
     nod=record
                nume:string;
                gen:byte;
                st,dr,tata:reper
             end;
var rad,p,q,pp:reper;
    nume:string;
    g1,g2:byte;
```

```
procedure creare(var p:reper;tata:reper;g:byte);
var nume:string;
begin
     readln(nume);
     if nume='' then p:=nil
                 else
                     begin
                          new(p);p^.nume:=nume;p^.tata:=tata;
                          p^.gen:=g;
                          write('Primul fiu al lui ',nume,':');
                          creare(p^.st,p,g+1);
                          write('Fratele lui ',nume,':') ;
                          creare(p^.dr,tata,g)
                       end
  end;

  function gasit(p:reper):boolean;
  var g:boolean;
  begin
       if p<>nil then
          if p^.nume=nume then
                       begin
                         gasit:=true;pp:=p
                       end
                          else
                       begin
                         g:=gasit(p^.st);
                         if g then begin gasit:=g;exit end;
                         gasit:=gasit(p^.dr)
                       end
                    else gasit:=false
  end;









  Begin
       clrscr;
       write('Stramosul: ');
       creare(rad,nil,0);
       repeat
             write('Persoana 1: ');
             readln(nume);
        until gasit(rad);
        p:=pp;
        repeat
              write('Persoana 2: ');
              readln(nume)
        until gasit(rad);
        q:=pp;
        g1:=p^.gen;  {g1=generatia pers. 1}
        g1:=q^.gen;  {g2=generatia pers. 2}
        if g1<g2 then
            repeat q:=q^.tata; dec(g2) until g1=g2
                 else
            if g1>g2 then
               repeat p:=p^.tata; dec(g1) until g1=g2;
```

```
          while p<>q do
                begin
                   p:=p^.tata; q:=q^.tata end;
          writeln('Stramosul comun cel mai apropiat este: ',p^.nume);
          readln
    end.
```

5. *Arbore de sortare:*

Sortarea unui sir de numere prin selectie arborescenta.

```
program arb_sortare;
type reper=^nod;
     nod=record
       inf:integer;
       st,dr:reper
     end;
var p:reper;
    a:array[1..20] of integer;
    n,x,y,dk,r,u,v,i:integer;


function min(x,y:integer):integer;
begin
     if x<y then min:=x else min:=y;
end;

procedure calcul(n:integer; var r,dk:integer);
begin
     dk:=1; r:=-1;
     while dk<n do
          begin
               dk:=dk*2;
               r:=r+1
          end
end;





procedure creare(var q:reper;niv:integer);
begin
     if niv<r then
        begin
             new(q);
             creare(q^.st,niv+1);
             creare(q^.dr,niv+1);
             q^.inf:=min(q^.st^.inf,q^.dr^.inf)
        end
           else if niv=r then
                begin
                     u:=u+1;
                     if u<=x div 2 then
                        begin
                             new(q);
                             creare(q^.st,niv+1);
                             creare(q^.dr,niv+1);
                             q^.inf:=min(q^.st^.inf,q^.dr^.inf);
                        end
                           else if y<>0 then
                                begin
```

```
                                        v:=v+1;
                                        new(q);
                                        q^.inf:=a[v];
                                        q^.st:=nil;
                                        q^.dr:=nil
                                   end
                  end
                          else begin
                               v:=v+1;
                               new(q);
                               q^.inf:=a[v];
                               q^.st:=nil;
                               q^.dr:=nil;
                               end
end;
procedure cautare(q:reper);
begin
     if q^.st=nil then q^.inf:=maxint
                  else if q^.inf<>maxint then
                     begin
                       if q^.st^.inf=q^.inf then cautare(q^.st)
                                            else cautare(q^.dr);
                       q^.inf:=min(q^.st^.inf,q^.dr^.inf)
                     end
end;
begin
     write('n=');readln(n);
     writeln('Dati sirul');
     for i:=1 to n do read(a[i]);
     calcul(n,r,dk);
     y:=dk-n;
     x:=n-y; u:=0;v:=0;
     creare(p,0);
     a[1]:=p^.inf;
     for i:=2 to n do
        begin
          cautare(p);
          a[i]:=p^.inf
        end;
     writeln('Sir ordonat');
     for i:=1 to n do write(a[i],' ');
end.
```

6. Forma poloneza a expresiilor aritmetice:

```
program forma_pol;
type reper=^nod;
     nod=record
       v:char;
       st,dr:reper
       end;
var n,i:byte;
    rad:reper;
    c:char;
    sir:array[1..30] of char;

procedure getchar;
begin
     c:=sir[i];
     i:=i+1;
end;
```

```
procedure factor(var r:reper);forward;
procedure termen(var r:reper);forward;
procedure putere(var r:reper);forward;


procedure exp(var r:reper);
var p:reper;
begin
     termen(r);
     p:=r;
     while c in ['+','-'] do
            begin
                 new(r);
                 r^.v:=c;
                 r^.st:=p;
                 termen(r^.dr);
                 p:=r;
            end
end;


procedure termen(var r:reper);
var p:reper;
begin
     factor(r);
     p:=r;
     while c in ['*','/'] do
           begin
                 new(r);
                 r^.v:=c;
                 r^.st:=p;
                 p:=r;
                 factor(r^.dr);
           end
end;









procedure factor(var r:reper);
var p:reper;
begin
     putere(r);
     p:=r;
     if i<=n then getchar;
     while c='^' do
           begin
               new(r);
               r^.v:=c;
               r^.st:=p;
               p:=r;
               putere(r^.dr);
               if i<=n then getchar;
           end
end;


procedure putere(var r:reper);
var p:reper;
begin
```

```
        getchar;
        if c='(' then exp(r)
                 else begin
                      new(r);
                      r^.v:=c;
                      r^.st:=nil;
                      r^.dr:=nil
                      end;
   end;

   procedure afisare(r:reper);
   begin
        if r<>nil then
           begin
                write(r^.v);
                afisare(r^.st);
                afisare(r^.dr);
                end
   end;

   begin
        write('Lungimea expresiei=');
        readln(n);
        writeln('Dati expresia: ');
        for i:=1 to n do read(sir[i]);
        i:=1;exp(rad);
        writeln('Forma poloneza prefixata: ');
        afisare(rad);
        readln
   end.
```

GRAFURI NEORIENTATE

1. Parcurgerea BF:

```
program parcurgere_BF;
var viz:array[1..20] of 0..1;
    A:array[1..20,1..20] of integer;
    n,m,p,i,j,x,y,u,v:integer;
    C:array[1..20] of integer;
begin
     write('n,m=');readln(n,m);
     for i:=1 to n do
         for j:=1 to n do A[i,j]:=0;
     for i:=1 to m do
         begin
```

```
            write('muchia ',i,'=');
            readln(x,y);
            A[x,y]:=1;A[y,x]:=1;
        end;
    write('varful de plecare : ');readln(i);
    for j:=1 to n do viz[j]:=0;
    C[1]:=i;
    p:=1;u:=1;
    viz[i]:=1;
    while p<=u do    {coada nu este vida}
          begin
               v:=C[p];
               for j:=1 to n do
                   if (A[v,j]=1) and (viz[j]=0) then
                      begin
                           u:=u+1;
                           C[u]:=j;
                           viz[j]:=1
                      end;
               p:=p+1
          end;
    write('lista varfurilor in parcurgere BF , plecand din ',i,' este: ');
    for j:=2 to n do write(C[j],' ');
    writeln;
    readln
    end.
```

2. Parcurgerea DF:

```
program parcurger_DF;
var viz:array[1..20] of 0..1;
    A:array[1..20,1..20] of integer;
    n,m,k,ps,i,j,x,y:integer;
    S,urm:array[1..20] of integer;

 begin
     write('n,m=');readln(n,m);
     for i:=1 to n do
         for j:=1 to n do A[i,j]:=0;
     for i:=1 to m do
         begin
              write('muchia ',i,'=');readln(x,y);
              A[x,y]:=1;A[y,x]:=1;
        end;
     write('varful de plecare: ');readln(i);
     for j:=1 to n do begin
```

```
            viz[j]:=0;
            urm[j]:=0
            end;
    S[1]:=i;ps:=1;
    viz[i]:=1;
    write('lista varfurilor in parcurgere DF plecand din ',i,' este:');
    while ps>=1 do
          begin
               j:=S[ps];
               k:=urm[j]+1;
               while(k<=n) and ((A[j,k]=0) or (A[j,k]=1) and(viz[k]=1)) do
                k:=k+1;
               urm[j]:=k;
               if k=n+1 then ps:=ps-1
                        else begin
                             write(k,' ');
                             viz[k]:=1;
                             ps:=ps+1;
                             S[ps]:=k;
                             end
          end;
          readln
end.
```

<u>3. Grafuri euleriene (</u> verificarea faptului ca un graf este eulerian):

```
program verif;
var eulerian:boolean;
    e,gr,c,c1:array[1..20] of integer;
    a:array[1..20,1..20] of integer;
    n,m,i,j,k,k1,x,y,l:integer;
    viz:array[1..20] of byte;

function vf_iz :boolean;  {val este true daca graful are vf izolate}
var i,j,s:byte;
begin
     vf_iz:=false;
     for i:=1 to n do
      begin
         s:=0;
         for j:=1 to n do s:=s+a[i,j];
```

```
          gr[i]:=s;
          if s=0 then
             begin
                  vf_iz:=true;
                  break;
             end;
        end;
  end;

  function grade_pare:boolean;  {are val true daca gradele tuturor}
  var i:byte;                     { varfurilor sunt numere pare}
  begin
       grade_pare:=true;
       for i:=1 to n do
          if odd(gr[i]) then
             begin
                  grade_pare:=false;
                  break
             end;
  end;

  function conex:boolean;  {are val true daca graful este conex}
  var b,viz:array[1..20] of byte;
      i,j,p,u,v:byte;
  begin
       for j:=1 to n do viz[j]:=0;
       b[1]:=1;
       p:=1;u:=1;viz[1]:=1;
       while p<=u do
             begin
                  v:=b[p];
                  for j:=1 to n do
                      if (a[v,j]=1) and (viz[j]=0) then
                         begin
                              u:=u+1;
                              b[u]:=j;
                              viz[j]:=1
                         end;
                  p:=p+1;
                  end;
       conex:=true;
       for i:=1 to n do
     if viz[i]=0 then
        begin
             conex:=false;
             break;
        end;
 end;

Begin
 write('n,m=');readln(n,m);
 for i:=1 to n do
     for j:=1 to n do a[i,j]:=0;
 writeln('dati muchiile grafului: ');
 for i:=1 to m do
     begin
          write('mucjia ',i,'=');readln(x,y);
          a[x,y]:=1;a[y,x]:=1;
     end;
 eulerian:=not vf_iz and grade_pare and conex;
 if not eulerian then writeln('graful nu este eulerian')
             else
                begin    {se construieste in C primul ciclu}
                   writeln('graful este eulerian');
                   c[1]:=1;
```

```
                    k:=1;
                    repeat
                       for j:=1 to n do
                           if a[c[k],j]=1 then
                              begin
                                   k:=k+1;
                                   c[k]:=j;
                                   a[c[k-1],j]:=0 ;
                                   a[j,c[k-1]]:=0;
                                   gr[j]:=gr[j]-1;
                                   gr[c1[k1-1]]:=gr[c1[k1-1]]-1;
                                   break;
                              end;
                   until c1[k1]=c1[1];
                   for j:=k downto l do c[j+k1-1]:=c[j];{se depl elem in C}
                   for j:=1 to k1-1 do c[j+l]:=c1[j+1];
                   k:=k+k1-1;
               end;
  writeln('ciclu eulerian: ');
  for i:=1 to m+1 do write(' ',c[i]);
  readln
 end.
```

3. ***<u>Grafuri hamiltoniene(</u>***se determina toate ciclurile hamiltoniene dintr-un graf):

```
program hamiltonian;
var x:array[1..10] of integer;
    a:array[1..10,1..10] of integer;
    m,n,i,j,k,v,sol,p,q:integer;

begin
     write('m,n=');readln(m,n);
     for p:=1 to n do
         for q:=1 to n do a[p,q]:=0;
     for k:=1 to m do begin
         write('muchia ',k,'= ');readln(p,q);
         a[p,q]:=1;a[q,p]:=1
         end;
     sol:=0;
     x[1]:=1;
     k:=2;x[2]:=1;
     while k>1 do
```

```
        begin
            v:=0;
            while (x[k]+1<=n) and (v=0) do
                begin
                  x[k]:=x[k]+1;
                  v:=1;
                  for i:=1 to k-1 do
                      if x[k]=x[i] then begin
                                   v:=0;
                                   break
                                   end;
                  if a[x[k-1],x[k]]=0 then v:=0;
                end;
            if v=0 then k:=k-1
                   else if (k=n) and (a[x[n],x[1]]=1)
                     then
                       begin
                        sol:=sol+1;
                        if sol=1 then writeln(' cicluri hamiltoniene :');
                        for j:=1 to n do write (x[j]);
                        writeln;
                       end
                     else if k<n then
                        begin
                          k:=k+1;
                          x[k]:=1;
                        end;
       end;
      if sol=0 then write('graful nu este hamiltonian');
      readln;
end.
```

4. ***<u>Algoritmul lui Dijkstra-</u>*** lanturi minime într-un graf neorientat :

```
program lanturi_minime;  {ALGORITMUL LUI DIJKSTRA}
const inf=65535;
var L:array[1..20,1..20] of word;  {L-lungimile muchiilor}
    d:array[1..20] of word;
      { d[i]= lungimea minima a lanturilor de la i0 la i}
    C:array[1..20] of boolean;
      {C= multimea varfurilor candidate la solutia la solutia S}
    vf_ant:array[1..20] of byte;
      {vf_ant[i]=anteriorul lui i pe lantul minim}
    i,j,i0,n,m:byte;
    dd:word;

procedure citire;
var i,j,k:byte;
    LL:word;
begin
     write('n,m,i0:');readln(n,m,i0);
```

```
     write('muchiile si lungimile lor:');
     for k:=1 to m do
         begin
              readln(i,j,LL);
              L[i,j]:=LL; L[j,i]:=LL;
         end
end;

procedure initializari;
var i,j:byte;
begin
     for i:=1 to n do d[i]:=inf;
     d[i0]:=0;
     for j:=1 to n do
         if L[i0,j]>0   {vf_ant j este adiacent cu i0 }
           then begin
                d[j]:=L[i0,j]; vf_ant[j]:=i0;
                end;
     for i:=1 to n do C[i]:=true;
     C[i0]:=false;  {i0 apartine lui S}
end;

procedure lant_minim(i:byte);
begin
     if i<>i0 then lant_minim(vf_ant[i]);
     write(i:3);
end;

function d_min(var i:byte):word;
var j:byte;   {d_min =min(d[j])  }
    min:word;
begin
     min:=inf;
     for j:=1 to n do if C[j] then
         if d[j]<min then begin min:=d[j]; i:=j end;
     d_min:=min;
end;





procedure scriere;
var i:byte;
begin
     for i:=1 to n do
         if i<>i0 then
            if C[i] then
               writeln('Nu exista lant de la ',i0,'la ',i)
               else begin
                    write('lantul minim de la ',i0, 'la ',i);
                    write('de lungime =',d[i]:2,' este:');
                    lant_minim(i);writeln
                    end
end;

begin
     citire; initializari;
     while d_min(i)<inf do
           begin
                C[i]:=false;
                for j:=1 to n do
                    if C[j] and (L[i,j]>0) then
                       begin
```

```
                        dd:=d[i]+L[i,j];
                        if dd<d[j] then begin d[j]:=dd;
                           vf_ant[j]:=i
                           end
                        end
     end;
     scriere
end.
```

5. Problema comis ñvoiajorului:

```
program comisv;
type natural=0..maxlongint;
var D:array[1..20,1..20] of word;
        {D-matricea distantelor dintre orase}
    c,cmin:array[1..20] of byte;
    neales:array[1..20] of boolean;
    k,n,oras_prec,oras:byte;
    dmin,dist:natural;

procedure citire;
var i,j,k:byte;
    dd:word;
begin
     write('n=');readln(n);
     for i:=1 to n-1 do
       for j:=i+1 to n do
           begin
             write ('distanta dintre orasele ',i,'si ',j,'=');
             readln(dd);
             D[i,j]:=dd;D[j,i]:=dd;
           end
end;









function oras_dist_min(i:byte):byte;
var j,jmin:byte;min:word;
begin
     min:=65535;
     for j:=1 to n do
         if neales[j] and (D[i,j]<min) then
            begin min:=D[i,j]; jmin:=j end;
         oras_dist_min:=jmin;
end;

function dist_ciclu(k:byte):natural;
var i:byte;{det euristic ciclul minim care pleaca din k}
begin
     for i:=1 to n do neales[i]:=true;
     c[1]:=k;
     neales[k]:=false;
     dist:=0;
     for i:=2 to n do
     begin
          oras_prec:=c[i-1];
```

```
            oras:=oras_dist_min(oras_prec);
            c[i]:=oras;neales[oras]:=false;
            dist:=dist+D[oras_prec,oras]
        end;
        dist:=dist+D[oras,k];
        dist_ciclu:=dist;
end;

begin
        citire; dmin:=maxlongint;
        for k:=1 to n do
            begin
              dist:=dist_ciclu(k);
              if dist< dmin then begin cmin:=c; dmin:=dist end;
            end;
        write('traseul minim are lungimea= ',dmin);
        writeln('si trece prin orasele :');
        for k:=1 to n do write(cmin[k]:3);
        writeln(cmin[1]:3);readln
end.
```

4. GRAFURI ORIENTATE

1. Algoritmul lui Dijkstra:

Sa se determine pentru toate v, rfurile xi ale unui graf orientat pentru care exista drum de la x0 la xi , lungimea celui mai scurt drum precum si unul dintre drumurile minime de la x0 la xi

```
program dijkstra;
const nmax=15;
    inf=maxint div 2;
var c:array[1..nmax,1..nmax] of integer;
    k,i,j,arc,m,n,x,y,z,xp:integer;
    s,d,prec:array[1..nmax] of integer;
    g:boolean;

procedure min( var k: integer);
var m,i:integer;
```

```
begin
     m:=inf*2;
     for i:=1 to n do
         if (s[i]=0) and (d[i]<m) then
            begin m:=d[i];
                  k:=i
            end
end;

procedure drum(i:integer);
begin
     if i<>0 then
        begin
             drum(prec[i]);
             write(i);
        end
           else writeln
end;

begin
 writeln('dati nr de noduri:');readln(n);
 for i:=1 to n do
     for j:=1 to n do c[i,j]:=inf;
 for i:=1 to n do c[i,i]:=0;
 writeln( 'dati nr de arce: ');readln(arc);
 for i:=1 to arc do
     begin
          write('dati arcul ',i,' si lungimea: ');
          readln(x,y,z);
          c[x,y]:=z;
     end;
 read(xp);
 for i:=1 to n do
  begin
     d[i]:=c[xp,i];
     s[i]:=0;
     if c[xp,i]< inf then prec[i]:=xp
                      else prec[i]:=0;
  end;
 s[xp]:=1;
 prec[xp]:=0;
 g:=true; x:=0;

  repeat
       min(k);
       x:=x+1;
       if (d[k]=inf) or (x=n)
          then g:=false
          else
              begin s[k]:=1;
                    for j:=1 to n do
                        if(s[j]=0) and(d[j]>d[k]+c[k,j]) then
                             begin
                                 d[j]:=d[k]+c[k,j];
                                 prec[j]:=k;
                             end;
              end;
 until (not g);
 for i:=1 to n do
  if i<>xp then
     if d[i]=inf then begin
        write('nu exista drum de la ', xp, 'la',i);
        writeln;
        end
           else begin
```

```
                    writeln('drum minim de la',xp,' la ',i);
                    drum(i);
                    writeln
                    end
end.
```

2. *Algoritmul lui Roy-Floyd -* determinarea tuturor drumurilor minime între oricare doua vârfuri ale unui graf :

```
program roymin;
const nmax=20;
      inf=maxint div 2;
      type multime=set of 1..nmax;
      var c:array[1..nmax,1..nmax] of word;
      {c-initial matricea drumurilor}
      d:array[1..nmax,1..nmax] of multime;
      dr:array[1..nmax] of 1..nmax;
      n,m,i,j,k,lg:word;

procedure initc; {initializeaza matricea costurilor C}
var i,j,x,y,z:word;
begin
     write('dati nr de noduri :');readln(n);
     for i:=1 to n do
         begin
              for j:=1 to n do c[i,j]:=inf;
              c[i,i]:=0
         end;
     write(' dati nr de arce: ');readln(m);
     for i:=1 to m do
         begin
              write('extremitatile si lungimea arcului ',i,':');
              readln(x,y,z);c[x,y]:=z;
         end
end;



procedure initd;
{initializeaza matricea D}
var i,j: integer;
begin
     for i:=1 to n do
       for j:=1 to n do
         if (c[i,j]<inf)and (i<>j) then d[i,j]:=[i]
            else d[i,j]:=[];
end;


procedure drum(i,j:integer);
{genereaza in vectorul dr un drum minim de la i la j pornind din nodul j}
var k:word;
begin
     if i<>j then
       begin
        for k:=1 to n do
         if k in d[i,j] then
            begin
                 lg:=lg+1;
```

```
                 dr[lg]:=k;
                 drum(i,k);
                 lg:=lg-1
            end;
        end
     else begin
          writeln;
          for k:=lg downto 1 do
              write(dr[k]:4)
          end
end;



procedure afisare;
var i,j:word;
{afisarea rezultatelor}
begin
     for i:=1 to n do
         for j:=1 to n do
           begin
           writeln;
           if c[i,j]=inf then writeln('nu exista drum intre',i,'si',j)
                 else
         begin
          writeln('lungimea drumurilor minime de la
',i,'la',j,'este',c[i,j]);
          if i<>j then begin
                  lg:=1;
                  dr[1]:=j;
                  drum(i,j)
                  end
          end
      end
end;









begin
     initc;
     initd;
     for k:=1 to n do
         for i:=1 to n do
           for j:=1 to n do
             if c[i,j]>c[i,k]+c[k,j] then
                begin
                  c[i,j]:=c[i,k]+c[k,j];
                  d[i,j]:=d[k,j];
                end
                  else
                    if c[i,j]=c[i,k]+c[k,j] then
                       d[i,j]:=d[i,j]+d[k,j];
     afisare;
end.
```

3. Determinarea tuturor drumurilor maxime între oricare doua noduri ale unui graf orientat:

```
program roymax;
const nmax=20;
      inf=maxint div 2;
      type multime=set of 1..nmax;
      var c:array[1..nmax,1..nmax] of word;
      {c-initial matricea drumurilor}
      d:array[1..nmax,1..nmax] of multime;
      dr:array[1..nmax] of 1..nmax;
      n,m,i,j,k,lg:word;

procedure initc; {initializeaza matricea costurilor C}
var i,j,x,y,z:word;
begin
     write('dati nr de noduri :');readln(n);
     for i:=1 to n do
         begin
              for j:=1 to n do c[i,j]:=inf;
              c[i,i]:=0
         end;
     write(' dati nr de arce: ');readln(m);
     for i:=1 to m do
         begin
              write('extremitatile si lungimea arcului ',i,':');
              readln(x,y,z);c[x,y]:=z;
         end
end;


procedure initd;
{initializeaza matricea D}
var i,j: integer;
begin
     for i:=1 to n do
       for j:=1 to n do
         if (c[i,j]>inf)and (i<>j) then d[i,j]:=[i]
            else d[i,j]:=[];
end;





procedure drum(i,j:integer);
{genereaza in vectorul dr un drum minim de la i la j pornind din nodul j}
var k:word;
begin
     if i<>j then
       begin
        for k:=1 to n do
         if k in d[i,j] then
            begin
                 lg:=lg+1;
                 dr[lg]:=k;
                 drum(i,k);
                 lg:=lg-1
            end;
        end
     else
         begin
          writeln;
          for k:=lg downto 1 do
```

```
              write(dr[k]:4)
         end
end;

procedure afisare;
var i,j:word;
{afisarea rezultatelor}
begin
 for i:=1 to n do
  for j:=1 to n do
   begin
    writeln;
    if c[i,j]=inf then writeln('nu exista drum intre',i,'si',j)
                   else
       begin
      writeln('lungimea drumurilor minime de la ',i,'la',j,'este',c[i,j]);
      if i<>j then begin
                  lg:=1;
                  dr[1]:=j;
                  drum(i,j)
                  end
       end
   end
end;

begin
     initc;
     initd;
     for k:=1 to n do
         for i:=1 to n do
           for j:=1 to n do
             if(c[i,k]<>inf) and (c[k,j]<>inf)then
               if c[i,j]<c[i,k]+c[k,j] then
                begin
                  c[i,j]:=c[i,k]+c[k,j];
                  d[i,j]:=d[k,j];
                end
                  else
                    if c[i,j]=c[i,k]+c[k,j] then
                       d[i,j]:=d[i,j]+d[k,j];
     afisare;
end.
```

4._Metoda drumului critic Óntr-un graf de activitati:

```
program drcritic;
const nmax=20;
      inf=maxint;
var l:array[1..nmax,1..nmax] of real;
      t,tb:array[1..nmax] of real;
      {in t se retin datele asteptate iar in tb datele limita de realizare
a evenimentelor}
      n,i,j:word;



procedure citire;
{citeste arcele si duratele asociate acestora}
var i,m,x,y:word;c:real;
begin
writeln('dati nr. de activitati:');
```

```
readln(m);
for i:=1 to m do begin
       writeln('dati arcul asociat activitatii nr.',i,' si durata
acesteia:');
       readln(x,y,c);
       l[x,y]:=c
       end;
end;

procedure calct(i:word);
{calculeaza recursiv elementul t[i]}
var max: real;j:word;
begin
     if i<2 then t[1]:=0
            else begin
                 max:=0;
                 for j:=1 to n do
                     if l[j,i]>=0 then
                        begin
                          if t[j]<0 then calct(j);
                          if max<l[j,i]+t[j] then
                             max:=l[j,i]+t[j]
                          end;
                     t[i]:=max
                 end
end;

procedure calctb(i:word);{calculeaza recursiv elementul tb[i]}
var min:real;j:word;
begin
     if i=n then tb[i]:=t[i]
        else begin
             min:=inf;
             for j:=1 to n do
                 if l[i,j]>=0 then
                   begin
                      if tb[j]<0 then calctb(j);
                      if min>tb[j]-l[i,j] then min:=tb[j]-l[i,j];
                   end;
                 tb[i]:=min
             end
end;



begin
     writeln('dati nr de evenimente ale lucrarii:');readln(n);
     for i:=1 to n do begin
         t[i]:=-1;
         tb[i]:=-1;
         for j:=1 to n do l[i,j]:=-1
         end;citire;
         calct(n);
         calctb(1);
         writeln('EVENIMENTELE CRITICE SUNT:');
         for i:=1 to n do
             if t[i]=tb[i] then writeln('evenimentul ',i);
         writeln('ACTIVITATILE CRITICE SUNT:');
         for i:=1 to n do
           for j:=1 to n do
             if (t[i]=tb[i])and(t[j]=tb[j])and(t[j]=t[j]+l[i,j]) then
             writeln('activitatea reprezentata de arcul :(',i,',',j,')');
end.
```

5. Desenarea arborilor binari cu un numar fixat de v, rfuri:

```
program arb_bin;
uses crt,graph;
type arbore=^varf;
     varf=record
       st,dr:arbore
       end;
var k,gd,gm,i,n:integer;
    a:arbore;
    c:char;
    s:array[ 0..20] of integer;

function earb(n1,n2:integer):boolean;
var i:integer;
begin
     earb:=false;
     if(n2=n1+2) and (s[n1]=1) and (s[n1+1]=2) and (s[n2]=2)
       then earb:=true
       else if (n2>n1+2) then
            if (s[n1]=1) and (s[n2]=2) and earb(n1+1,n2-1)
                  then earb:=true
                  else if(s[n1]=1) and (s[n1+1]=2) and earb(n1+2,n2)
                        then earb:=true
                        else for i:=n1+3 to n2-3 do
                           if (s[n1]=1) and earb(n1+1,i) and earb(i+1,n2)
                                then begin
                                     earb:=true;
                                     exit
                                     end
end;





procedure afisare(a:arbore;x,y:integer);
begin
     if a<>nil then
        begin
             if a^.st<>nil then
                begin
                     line(x,y,x-30,y+30);
                     afisare(a^.st,x-30,y+30)
                end;
                if a^.dr<>nil then
                   begin line(x,y,x+30,y+30);
                         afisare(a^.dr,x+30,y+30)
                   end
        end
end;

function constr(n1,n2:integer):arbore;
var i:integer;
    a:arbore;
begin
     if (n2=n1+2) and (s[n1]=1) and (s[n1+1]=2) and(s[n2]=2)
```

```
        then begin
             new(a);a^.st:=nil;
             a^.dr:=nil;constr:=a
             end
        else if (n2>n1+2)
             then if (s[n1]=1) and (s[n2]=2) and earb(n1+1,n2-1)
                  then begin
                       new(a); a^.dr:=nil;
                       a^.st:=constr(n1+1,n2-1);
                       constr:=a
                       end
                  else if (s[n1]=1) and (s[n1+1]=2) and earb(n1+2,n2)
                          then begin
                               new(a);a^.st:=nil;
                               a^.dr:=constr(n1+2,n2);
                               constr:=a
                               end
                          else for i:=n1+3 to n2-3 do
                            if (s[n1]=1) and earb(n1+1,i) and earb(i+1,n2)
                                  then begin
                                       new(a);
                                       a^.st:=constr(n1+1,i);
                                       a^.dr:=constr(i+1,n2);
                                       constr:=a;exit
                                       end
end;

procedure stergere(var a:arbore);
begin
     if a<>nil then
        if (a^.st=nil) and (a^.dr=nil)
         then
              begin dispose(a);a:=nil end
         else
              begin
                   stergere(a^.st);
                   stergere(a^.dr);
                   stergere(a)
              end
end;




begin
     clrscr;
     writeln('numarul de varfuri :');readln(n);
     detectgraph(gd,gm);
     initgraph(gd,gm,'c:\bp\bgi');
     for i:=0 to 2*n do s[i]:=0;
     s[2*n+1]:=2;k:=0;
     while k>=0 do
           if k=2*n then
              begin
                   if earb(1,2*n+1) then
                      begin
                           a:=constr(1,2*n+1);
                           afisare(a,300,0);
                           repeat until keypressed;
                           c:=readkey;
                           cleardevice;
                           stergere(a)
                      end;
                   dec(k)
              end
```

```
                else if s[k+1]<2
                        then
                            begin
                             inc(s[k+1]);
                             inc(k)
                            end
                        else
                            begin
                                 s[k+1]:=0;
                                 dec(k)
                            end;
          closegraph;
     readln
end.
```

BIBLIOGRAFIE

1. **Doina Rancea**
   Limbajul Pascal- Algoritmi fundamentali
   Computer Libris Agora 1999

2. **Cornelia Ivasc - Mona Pruna**

Bazele informaticii- Grafuri si Elemente de Combinatorica
Editura Petrion - Bucuresti 1995

3. **Octavian Patrascoiu, Gheorghe Marian, Nicolae Mitroi**
Elemente de grafuri si combinatorica. Metode, algoritmi si programe
Editura ALL Bucuresti- 1994

4. **Valentin Cristea, Irina Athanasiu, Eugenia Kalisz, Valeriu Iorga**
Tehnici de programare
Editura Teora- Bucuresti-1997

5. **Knut D.E.**
Tratat de programarea calculatoarelor. Algoritmi fundamentali
Editura Tehnica- Bucuresti-1974

6. **Ghe. Barbu, Ion Vaduva**
Bazele Informaticii
Editura Tehnica-Bucuresti- 1997

7. **Ioan Tomescu**
Data Structures
Editura Univ. Bucuresti- 1997

8. **Tudor Sorin**
Tehnici de programare-1996